NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

EVALUATION OF HARDWARE AND SOFTWARE FOR A SMALL AUTONOMOUS UNDERWATER VEHICLE NAVIGATION SYSTEM (SANS)

by

Nancy Ann Norton

September 1994

Thesis Advisors: Robert McGhee
James Clynch

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	September 1994	Master's Thesis

4. TITLE AND SUBTITLE
EVALUATION OF HARDWARE AND SOFTWARE FOR A SMALL AUTONOMOUS UNDERWATER VEHICLE NAVIGATION SYSTEM (SANS) (U)

5. FUNDING NUMBERS

6. AUTHOR(S)
Norton, Nancy Ann

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the United States Government.

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release, distribution is unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

The purpose of this thesis is to evaluate the hardware and software for a Small Autonomous Underwater Vehicle (AUV) Navigation System (SANS), a self-contained, externally mountable navigation system. The SANS is designed to determine the location of an underwater object using a combination of Global Positioning System (GPS) while surfaced, and Inertial Navigation System (INS) while submerged.

Various experimental testing of the hardware was performed to determine the ability of the GPS navigation system to function within the mission requirements. A test was done to determine the time required to obtain a GPS fix. A test of the system while the antenna was covered with water was done to determine if the GPS signal could penetrate a shallow layer of water. Finally, a sea test was done to determine the feasibility of reacquiring a GPS fix after the system has been submerged during normal ocean wave wash. A computer simulation was written in Common LISP Object System (CLOS) in order to evaluate the errors introduced by using an accelerometer in the INS to determine the climb angle of the AUV while surfacing.

The experimental testing of the GPS system showed that the GPS signal is able to penetrate a shallow layer of water covering the antenna, and that the system is able to meet the accuracy and time requirements of the mission while being splashed by wave wash. The simulation results show that the error introduced by measuring climb angle with an accelerometer is minor and will not significantly degrade the accuracy of the system.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Autonomous Underwater Vehicle (AUV), Global Positioning System (GPS), Inertial Navigation System (INS), Submerged Navigation, SANS, Simulation, CLOS	153
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

Approved for public release; distribution is unlimited

EVALUATION OF HARDWARE AND SOFTWARE FOR A SMALL AUTONOMOUS UNDERWATER VEHICLE NAVIGATION SYSTEM (SANS)

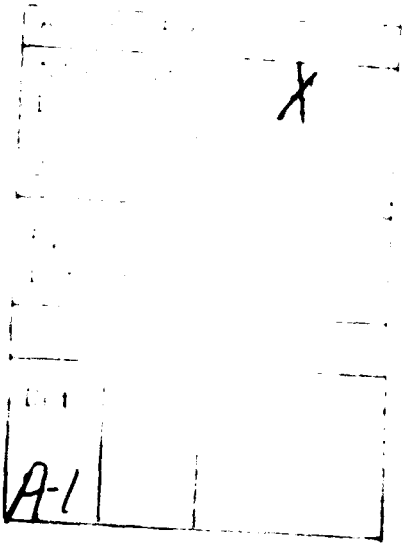
Nancy A. Norton
Lieutenant, United States Navy
B.S., Portland State University, 1986

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
September 1994



Author: Nancy A. Norton
Nancy A. Norton

Approved By: Robert B. McGhee
Robert McGhee, Thesis Advisor

James R. Clynch
James Clynch, Thesis Advisor

Ted Lewis
Ted Lewis, Chairman,
Department of Computer Science

ABSTRACT

The purpose of this thesis is to evaluate the hardware and software for a Small Autonomous Underwater Vehicle (AUV) Navigation System (SANS), a self-contained, externally mountable navigation system. The SANS is designed to determine the location of an underwater object using a combination of Global Positioning System (GPS) while surfaced, and Inertial Navigation System (INS) while submerged.

Various experimental testing of the hardware was performed to determine the ability of the GPS navigation system to function within the mission requirements. A test was done to determine the time required to obtain a GPS fix. A test of the system while the antenna was covered with water, was done to determine if the GPS signal could penetrate a shallow layer of water. Finally, a sea test was done to determine the feasibility of reacquiring a GPS fix after the system has been submerged during normal ocean wave wash. A computer simulation was written in Common LISP Object System (CLOS) in order to evaluate the errors introduced by using an accelerometer in the INS to determine the climb angle of the AUV while surfacing.

The experimental testing of the GPS system showed that the GPS signal is able to penetrate a shallow layer of water covering the antenna, and that the system is able to meet the accuracy and time requirements of the mission while being splashed by wave wash. The simulation results show that the error introduced by measuring climb angle with an accelerometer is minor and will not significantly degrade the accuracy of the system.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS

The author wishes to thank Dr. Robert McGhee and Dr. James Clynch who served as co-advisors for this thesis. Dr. McGhee's knowledge, guidance and enthusiasm for this project where instrumental in the success of this research. His assistance and instruction were vital to my understanding of all aspects of this research. Dr. Clynch provided a tremendous amount of technical information and knowledge outside of my field, as well as guidance in proper research procedures. His extensive time and effort spent with the author definitely improved the overall quality of this thesis. Both advisors made the entire project a very worthwhile and enjoyable learning experience.

My appreciation goes to Dr. Se-Hung Kwak for initially generating my interest in this research, and guiding me into a focus area for this thesis. He was always encouraging, and his wide range of knowledge and expertise were always available.

A great deal of gratitude goes to Mr. Russ Whalen, who provided the technical support and knowledge for this research. Without the time and labor he provided, this research would not have been possible.

The basis for this research was provided by LCDR Clark Stevens and LT James McKeon. Their effort and the knowledge they presented previously were essential to this work, and is greatly appreciated.

Finally, the author would like to thank her husband, LT Bruce Hamilton, for the selfless support and encouragement that he continually provides. His patience, understanding, and support make all of my work easier and more worthwhile.

I. INTRODUCTION

A. BACKGROUND

The mission of Autonomous Underwater Vehicles (AUVs) varies greatly. One of the most important and most difficult areas of AUV research is navigation. There are many types and combinations of navigation that can be used by AUVs. The type of navigation system that is desired depends largely on its mission. The Global Positioning System (GPS) provides a highly accurate means of navigating while on the surface. Various types of Inertial Navigation Systems (INSs) can be used for underwater navigation.

The Small AUV Navigation System (SANS) is designed for use on a variety of AUVs. These could include mechanical AUVs of varying sizes, humans, or marine mammals. The mission of the AUV equipped with a SANS is underwater mapping. This mission is broken down into two phases. After being launched from shore or from a boat, the first phase of the mission is the transit phase. This would include navigating the AUV from the launch site to the desired mission area, and navigating back to a designated recovery site. The second phase of the mission is the mapping phase. This involves searching a designated area for objects of interest and determining their locations.

In order to provide accurate navigation and mapping, a combination of GPS and INS navigation is used in the development of the SANS. The feasibility of this combination was evaluated favorably in [MCKE92]. The hardware and software architecture required to perform the mapping phase of the mission were defined and evaluated in [STEV93].

B. THE RESEARCH QUESTIONS

The research questions of this thesis are:

- How will the ocean surface and salt-water affect the ability of a GPS antenna and receiver to continue tracking satellites used by the SANS to determine location?
- What current technology and equipment are proposed for use in the production of a prototype SANS?

- Will the current SANS design and hardware meet all of the performance objectives for this mission?

C. SCOPE, LIMITATIONS AND ASSUMPTIONS

This thesis reports the findings of the third year of research in an ongoing research project. The scope of this investigation is to determine how the ocean water will effect the ability of the SANS to use GPS navigation with an antenna on the surface of the water, and to develop a computer simulation written in LISP, that will determine the error introduced by using an accelerometer in the SANS as part of the Inertial Navigation System (INS). The objectives for this project are described by McGhee et al. [MCGH92] as:

- Low Power Consumption. Operation from an external battery pack for 24 hours is desirable.

- Limited exposure time. GPS antenna exposure time in the mapping phase should be minimized. Up to 30 seconds exposure allowed, but intervals between such exposures should be as long as possible, exceeding several minutes at a minimum.

- Maintain covert operation. The GPS antenna should present a very small cross section when exposed and should not extend more than a few inches above the surface.

- Maximize accuracy. For the mapping phase of the mission, system positioning accuracy of 10 meters rms or better is required with post-processing, both submerged and surfaced.

- Minimize size. Total volume is not to exceed 120 cubic inches. An elongated, streamlined package is preferred.

D. ORGANIZATION OF THESIS

This thesis provides an evaluation of the hardware and software used in a low cost version of SANS called the *interim* SANS.[STEV93] No changes have been made to the software system in this research, however the software system is described. Changes have been made to the hardware system which have been evaluated, as well as additional testing

of the system that demonstrate its ability to operate in the ocean, based on the mission requirements. This thesis is organized into seven chapters.

Chapter II reviews the previous work on this project as well as previous and current work on GPS navigation, AUV submerged navigation, and robot kinematics and dynamics simulation.

Chapter III is a detailed problem statement. This includes the mission requirements, and the problems related to GPS navigation, submerged navigation, and the LISP navigation simulation.

In Chapter IV is a detailed description of the hardware and software currently in use for this project. It explains the hardware components currently used in the SANS and includes a description of their capabilities. Photographs of each piece of hardware are included. Chapter IV also describes the software designed and detailed in Stevens [STEV93].

Chapter V is a description and presentation of the results of the demonstration system testing performed, and an assessment of the system's ability to achieve the mission requirements.

Chapter VI is a description of the LISP simulation code used to determine expected error rates of the measurement of horizontal distance travelled. This includes an explanation of the class and object hierarchy used, and a sample display from a simulated AUV surfacing.

Chapter VII presents the conclusions of the software and hardware testing and provides recommendations for future research.

II. SURVEY OF PREVIOUS WORK

A. INTRODUCTION

Autonomous Underwater Vehicles (AUVs) have a potential for use in many different applications. One very important aspect of AUV control is navigation. Many potential applications require highly accurate navigation. This chapter will discuss some of the various possible solutions for navigating an AUV, and how robot kinematics and dynamics can be used for computer simulation of the actions of the AUV and therefore the requirements for navigation systems. The types of navigation can be split into two categories, external signal based navigation and sensor based navigation.

External signal based navigation provides positioning information only when the signal receiver is exposed to the signals, above the ocean surface, such as Loran, Omega and the Global Positioning System (GPS). Loran and Omega both are relatively inaccurate, and Loran provides limited coverage. Because GPS provides continuous worldwide coverage and is highly accurate, it will be the focus of this survey.

Sensor based navigation constitutes a self-contained system which can be made up of a wide variety of equipment that can be used to determine the submerged location of an object. These can be grouped into four types: dead reckoning, inertial navigation systems (INS), acoustic navigation, and geophysical mapping comparison. Sensors are subject to drift. On long AUV missions they are not able to provide the accuracy required for many applications. High quality INS are expensive as are acoustic beacons. Acoustic beacons must be predeployed at precisely known locations, and geophysical map interrogation requires a good bottom elevation map previously stored on the AUV's computer. [TUOH93]

B. GPS NAVIGATION

The Navigation Satellite Timing and Ranging Global Positioning System or NAVSTAR GPS is a United States Department of Defense project that was started in the 1970's to attempt to provide the military with precise navigation capability and accurate

timing [PARK80] and [BOSS85] It is a space based radio positioning navigation and time-transfer system The fully operational GPS system now provides 24 hour all weather navigation capability, by providing total earth coverage using 24 satellites orbiting the earth in circular 20,200 km orbits at a 55° inclination with 12-hour periods They operate on two L-band frequencies, L1 (1575.4 MHz) and L2 (1227.6 MHz) Superimposed on these two carrier frequencies is navigation and system data, predicted satellite ephemeris, or position information, atmospheric propagation correction data, satellite clock error information, and satellite health data. [VAND80] and [WOOD85]

There are two unique and distinct navigation services available from the GPS satellites, the Standard Positioning Service (SPS) and the Precise Positioning Service (PPS). In order to degrade the accuracy of SPS, an intentional inaccuracy is introduced into the satellite broadcast signal through a process called Selective Availability or SA. [KREM90] This limits SPS to worldwide coverage with 100 m horizontal accuracy with a 95% confidence level. PPS access is controlled by the use of cryptographic devices which remains restricted to US and allied military and highly selective, specific US non-military uses that are in the national interest. PPS provides the highest stand alone accuracy, 15 m SEP (Spherical Error Probable), a velocity accuracy of 0.1 m/sec, and a timing accuracy of better than 100 nanoseconds.[VAND80] and [WOOD85]

To take full advantage of the high accuracy potential of GPS, without the cryptographic equipment required for access to the P-code, the civilian community has had to determine a way of filtering the errors and improving the accuracy of the SPS. The solution was differential GPS or *DGPS*. DGPS uses one or more stationary antennas and one moving antenna to yield position with a much higher accuracy. This simply entails placing a receiver at a known location, determining the position, or pseudo range, errors and broadcasting the corrections to nearby users. [KREM90] Real-time differential processing and differential post-processing can improve GPS accuracy to 2-4 m with SA on or off, without the need for PPS [CLYN92] and [COCO90].

Over the last several years, GPS has been adapted for use in virtually every type of transportation in the military and the civilian world. The cost and the size of GPS antennas

and receivers have decreased significantly, making practical new uses that had only been concepts in the past. GPS is now used heavily on aircraft and nearly every size and type of ship. It is now even being used in automobiles and hand-held units are used by backpackers. [KRAU93]

Given the level of accuracy provided by GPS and the great decreases in size and cost of receivers, GPS is an obvious choice for use in AUV navigation. Many researchers have begun to include the capabilities of GPS into their applications. In [YOUN91], Youngberg suggests that free-roving buoys be used to translate GPS radio-based signals to an underwater acoustic-based signal. These sonobuoys (123 mm diameter x 910 mm long, 5-15 kg) would contain a GPS antenna, GPS receiver, processing and control subsystem, acoustic transmitter, battery power, and homing beacon. The AUV could then determine its position based on the ranging and position fixing of the buoys. A simulation of this type of system was done with a 1 km distance between buoys and a couple of hundred meters distance to the AUV. It is proposed that an accelerometer, a Kalman filter, and DGPS be used to overcome the errors caused by the ocean waves and variations in altitude. Each GPS buoy would broadcast its position data via spread spectrum signals used by the AUV for ranging. This would save much of the time and effort that must be spent to position and calibrate transducers. [LEU93]

Another buoy system, the NAVSYS TIDGET, was developed as a low-cost, disposable GPS for military sonobuoys that are air-launched and float on the surface. The design requirements were:

- Cost < $150
- 10 m accuracy
- Sized to fit within existing sonobuoys
- Antenna to fit in the sonobuoy float bag
- Time To First Fix nearly instantaneous
- No modification to the airborne acoustics processor
- Operates in all sea states
- Maximum update rate of 10 seconds

- No pre-launch initialization requirement
- Power < 1 W continuous

The TIDGET was able to meet all of the design requirements. [BROW93]

Another area of continuous improvement is miniaturization. GPS receivers have been able to maintain or even improve in performance while getting much smaller. For example, the Furuno GPS Receiver Module LGN-72 is an eight-channel receiver which is a single printed circuit board that requires only 2 W of power and is 100 mm x 70 mm x 20 mm. [SOUE92]

C. AUV SUBMERGED NAVIGATION

Dead reckoning in some form has been used for navigating longer than any of the other methods. Position is calculated by integrating the velocity of the vehicle in time. Velocity can be determined simply by using a compass for heading measurement and tracking speed. Modern dead reckoning systems typically use magnetic or gyroscopic heading sensors and a bottom or water locked velocity sensor. [GROS92] Errors are introduced by the unknown movement of the vehicle caused by ocean currents and waves.

There are many ways to calculate position while submerged. Velocity measurements are available from Doppler sonar, or from correlation velocity log (another type of sonar). Position can be measured directly by laser scanning, CCD cameras, side scan sonar, and magnetic field measurements. Position can also be inferred by double integration of acceleration as in Inertial Navigation Systems (INS). CCD cameras operating close to the bottom may be used to recognize features and compare them to a reference map. Side scan sonar measures variations in the ocean bottom backscatter. [BERG93] INS determine position by a combination of precision orientation or angular rate sensors and accelerometers. There are many different varieties of angular sensors and accelerometers. These include using fiber optic gyros, ring laser gyros, vibratory rate sensors, high performance Inertial Measurement Units (IMUs) with three inertial grade angular rate sensors and three precision linear accelerometers, and three-axis magnetometers with two-

axis inclinometers. All of these sensors are subject to drift errors however, and the error increases with time. Also, high quality sensors are very expensive.

Acoustic navigation uses the time of arrival and direction of the acoustic signal relative to an array of transponders to determine position. There are long, short, and ultrashort baseline systems (LBL, SBL and USBL). [TUOH93] For LBL, the acoustic beacons must be predeployed at a position that is known to an accuracy greater than that desired for the AUV. Long baseline navigation has been tested [BELL92] in which the AUV passively listens to the synchronized emissions of the array, where the transponder locations are known. The transponders are separated by 100 m to 10 km, depending on the frequency of operation used by the system. The vehicle uses the differential time of arrival of the acoustic pulses to determine its position. The advantages of the LBL system are that it can be used by multiple vehicles at one time, the hardware is fairly simple and small, and it requires minimal power usage by the AUV.

This system is being researched for use in Arctic under-ice surveying with transits of up to 10 km and mapping operations over an area of 1 km^2. [BELL93-2] Another application which uses three forms of navigation is the Odyssey AUV [BELL93-1] using dead reckoning, LBL to determine the position relative to the transponder array, and ultra short baseline which is used to track the AUV from the surface. A similar proposal is analyzed in [ROER93] where it is proposed to use an AUV to collect data under oil spills, at a depth of 1-3 m. A boom floating on the surface is used to surround the oil spill and the AUV would have to maintain a position relative to this boom. The proposal is to place three acoustic transponders on the boom, and the AUV would navigate relative to the transponders. Error would be introduced due to the movement of the boom and acoustic multipath and water column thermoclines.

Geophysical mapping comparisons may be achieved using various techniques. They take measurements of simple geophysical properties with vehicle mounted sensors and match those properties to computer models on board the vehicle. [TUOH93] These geophysical properties may be magnetic field intensity, bottom topography or a known image of a structure. The sensors used are either sonar or acoustic transponders. Some

examples of sonar mapping include the use of multibeam sonar for a sea floor profile, which measures the depths at different angles to get an accurate profile of the sea floor [BERG93], use of high frequency sonars which yield resolution which can detect objects in the 50 to 100 m range [CRIS93], and EDOs new doppler precision velocity sensor which uses the latest digital signal-processing techniques and includes a completely new planar transducer array. [JORG94]

The Brimingham Acoustic Signalling System (BASS) testing was described in [COAT93]. This used a differential phase shift keyed, line-of-sight, data link which successfully transmitted over distances to 250 m. They used a 1/50 scale model of an underwater oil rig, and placed the model transducer within the model rig. The resulting data was then used to run a database search of the possible positions in and around the structure. They found that it is possible to determine the position of the model transducer to an accuracy equal to the quantization cell size of the map stored in the database. This does not provide direction of motion however. A two-tier position estimation procedure is proposed which would use both macro- and micro-navigation. [TUOH93] The macro-navigation is a coarse-scale position estimation via a logic-based interval filter and multiple hypothesis tracking. The micro-navigation is a fine-scale, quantitative position determination through interrogation of low-level physical property models which represent a continuum of possible locations. This approach is similar to the terrain contour matching guidance system of cruise missiles. [CZES93] and [HINR76] The drawback to any type of geophysical mapping comparison is that a good map must already be available and be stored in the computer database.

A combination of various systems has the potential to greatly reduce the disadvantages of any one system alone. Velocity aided navigational systems offer greater precision and less cost, and reduce the complexity and stability requirements of an INS. A precision velocity sensor can be used to provide drift compensation to an INS. The correlation sonar illuminates the ocean bottom to get a characteristic sonar signature. Then, the correlation velocity log (CVL) [GROS92] provides a method for determining absolute velocity relative to the ocean floor at a stand off distance much greater than other velocity

measuring techniques. A miniature AUV has been successfully used under Artic ice, with a dead reckoning system, comprised of a fluxgate compass, pressure transducer, pitch/roll sensor, yaw-rate gyro, and an acoustic homing system. [LIGH93] and [OSSE93] Theseus uses a combination of dead-reckoning using a medium accuracy INS, a doppler sonar, and an acoustic positioning system. [BUTL93]

McKeon proposed a combination of GPS and INS to allow an AUV to determine position information while submerged, and to surface in order to re-initialize the sensors with the high accuracy of the GPS. [MCKE92] This concept has received much attention. Early computer simulations using GPS to correct INS errors showed GPS estimated errors of 43 m. [NAGE92] An extended Kalman filter has been adapted for an AUV to optimize INS navigation. [MILL91] Several combination GPS/INS systems have been proposed and developed. Litton is now producing two completely solid-state INUs which use navigation quality fiber optic gyros (FOG) and micromachined silicon accelerometer (SiAc) technology [COX94]. Another is the Honeywell H-764G Integrated Embedded GPS/INS, which uses a ring laser gyro (RLG) based strapdown INS. This system has demonstrated pure inertial performance of less than 0.8 nmi/hr and GPS aided performance of less than 16 meters SEP. The combined system provides outstanding performance with fewer than four satellites in view, is reduced in size, weight and power requirement, and is simple to install. [MOYA93]

Two submersibles that are designed to surface to transmit data and establish a GPS position to update onboard guidance systems are DOLPHIN and DOGGIE. [HADD93] DOLPHIN (Deep Ocean Long Path Hydrographic Instrument) will navigate across the oceans conducting a continuous series of hydrographic measurements. It is intended to undulate between the surface and the ocean floor, surfacing at intervals of about 30 km to receive GPS position updates. The DOGGIE (Deep Ocean Geological and Geophysical Instrumented Explorer) is designed to investigate at a much finer resolution, features that have been identified initially with a sonar survey. It would have a 3-5 day mission of about 500 m above the sea floor and an area of 50 km x 50 km. The DOGGIE would also surface periodically and update the drift error in its dead reckoning sensors. The designers of these

two systems found that "...it is possible to reconstruct the GPS data message despite periodic signal outages...". An analysis of various types of INS systems can be found in [MCGH92] Various combinations of GPS/INS systems are available that can either be designed by the user or bought off-the-shelf. These combinations are able to provide very accurate, high rate, position and velocity information which doesn't degrade over time [BROW92-1].

D. ROBOT KINEMATICS AND DYNAMICS SIMULATION

Robot kinematics is a systematic approach of using vector/matrix algebra to represent the spatial geometry of an object with respect to a fixed frame of reference and as a function of time. Robot dynamics uses the mathematical equations and physical laws describing motion such as Newton-Euler equations to represent motion from applied forces and moments. [FU87] Computer simulation of robot motion is a very useful tool. Simulation can be used to determine desired motions, to analyze motions, and to determine the results of motions. The motion of rigid bodies, as described in physics and engineering, can be used to simulate movement to a desired location. [DAVI93] Forward kinematics and dynamics compute the Cartesian space position and orientation of a rigid body from the parameters that describe the body. [FU87] Davidson provides a thorough explanation of the kinematics and graphical computer simulation code for the Aquarobot, an underwater walking machine, written in Common Lisp Object System (CLOS) and C++. The object-oriented programming approach used in this work allows for the design of a complex system made of many components that may be governed by different physical laws. [DAVI93] This allows the programmer to easily make adjustments and modifications to individual components of the system in order to determine the effects of changes to the system being modeled.

E. SUMMARY

As shown in this survey, there are a multitude of possibilities that can be used for navigating an AUV. The choices range from simple dead reckoning to complex systems

that combine GPS and INS or GPS and acoustic navigation. The type of navigation system desired for a particular mission must be determined after analyzing the expected use of the AUV. A navigation system designed for one AUV may be extremely successful, but it may not adapt well to an AUV of a different size or with a different mission and with different requirements for accuracy. Because the range of possible systems is so broad, a computer simulation of an AUV with known mission parameters is a useful technique for determining an optimum navigation system design.

III. DETAILED PROBLEM STATEMENT

A. INTRODUCTION

As stated in Chapter II, there are many possible systems that could be used for AUV navigation. The role and mission of the AUV will greatly influence any navigation system design. The research of this thesis was done in support of an AUV with a primary mission of covertly mapping the location of objects of interest underwater. The AUV will be launched either from shore or from a boat, and will search a particular area for specific objects. Because the mission is covert, the search area may be a considerable distance from the launch area. Therefore the mission is broken up into two phases. The first phase is the transit phase, which will begin at the launch site and continue to the mapping location, and possibly between multiple mapping locations, then back to a recovery location. The AUV may be surfaced or intermittently submerged during the transit phase. The second phase is the mapping phase. The mapping phase of the mission will consist of repeated dives by the AUV to locate the object or objects of interest. An experimental AUV designed for this type of mission is described in [HEAL92].

The Small Autonomous Underwater Vehicle Navigation System (SANS) was proposed as a way to determine the precise location of any objects found by the AUV [MCGH92]. A GPS/INS combination was proposed for the SANS [MCKE92] in order to attempt to meet the research objectives described in [MCGH92]. The SANS was designed as an externally mounted, self-contained navigation system which could be used on a variety of AUVs for mapping the locations of objects found underwater. The specification for SANS requires that total volume not exceed 120 cubic inches and the package be in an elongated shape. The system must provide positioning accuracy of 10 meters rms or better with post-processing, while submerged and surfaced. In order to avoid detection, the GPS antenna must not extend more than a few inches out of the water, and must have a small cross section. Also, the antenna exposure time during each surfacing should be minimized to 30 seconds or less, with intervals between exposures of at least several minutes. Finally, the system must be able to operate from an external battery pack for up to 24 hours.

An interim SANS was designed and tested on land by Stevens [STEV93]. This is an interim design because it lacks the ability to locate multiple objects before surfacing to update the SANS sensors. The proposed solution with the current hardware is to require the AUV to surface after each object is located, obtain a GPS fix for updates, then continue the mapping phase. This is referred to as a "pop-up" maneuver. With improved system components, the need for this pop-up maneuver should be eliminated during the mapping phase. A proposed final SANS design called the *baseline* system [MCGH92], would contain those improved components.

Stevens found that by using differential post-processing, a GPS/INS combination using miniature gyroscopes and other commercially available equipment can provide the accuracy as well as the minimum exposure time required by the mission within the cost constraints of the project. However, this system was subject to a limitation on maximum climb or dive angle to prevent gyro tumbling. [STEV93] As a continuing evaluation of that interim system, this thesis is focused on experimental testing to determine its ability to operate and meet system objectives on and under water, and a computer simulation to determine the accuracy of a system that replaces the gyroscopes with accelerometers, to both reduce cost and eliminate restrictions on climb or dive angle.

B. GPS NAVIGATION

GPS navigation is able to provide highly accurate, worldwide positioning data very quickly. As described in the preceding chapter, the two positioning services available are the Standard Positioning Service (SPS) which is available to everyone, and the Precise Positioning Service (PPS) which is encrypted for restriction to US and allied military. Using SPS, the accuracy of the positioning information is limited to approximately 100 meters horizontal accuracy [VAND80]. PPS could be used for navigation systems on military applications such as this project. However, as shown in Table 1, using differential

GPS eliminates any need or desire to incur the potential risk involved in having cryptographic keys in the navigation system of an AUV.

POSITIONING SERVICE	PPS	SPS
NON-DIFFERENTIAL	16	100
DIFFERENTIAL	2-4	2-4

Table 1. GPS Positioning Accuracy (in meters) after [STEV93]

GPS navigation and a magnetic compass will provide the primary source of navigation data while in the transit phase of the AUV mission. The AUV must be able to submerge for short intervals and still maintain accurate navigation after resurfacing. The underwater mapping phase of the mission will require both surfaced and submerged navigation. The INS sensors used in the SANS are constrained by size and weight. The interim SANS has an additional cost constraint. Therefore the quality of the sensors used will require frequent updates in order to maintain adequate accuracy. The AUV will be required to surface in the area of the object found (perform a pop-up maneuver) to obtain a GPS fix, which will be used to record the current position and, with post-processing, extrapolate this position backwards to the submerged object.

The GPS receiver chosen for use in this research is a Motorola PVT-6 which can simultaneously track up to six GPS satellites. It is capable of providing position accuracy of better than 25 meters, SEP without Selective Availability (SA) on, 100 meters with SA on, 0.1 meter/second velocity accuracy, and a Time-To-First-Fix (TTFF) of about a minute, with reacquisition time in less than 4 seconds when the antenna has been obscured for 60 seconds. [MOTO93-2] There are different power-up states for GPS receivers, and the TTFF varies depending on the power-up state. This is discussed further in the description of the Motorola PVT-6 in Chapter IV, and the complete specifications are listed in Appendix A. This receiver is capable of providing positioning data within the accuracy and time requirements of this project, under normal operating conditions [STEV93]. A new series of tests was performed to determine the GPS capabilities of the receiver when interacting with sea water. The results obtained are also presented in Chapter IV.

In order to avoid detection, the antenna for the navigation system must be as unobtrusive as possible, from the air, water and land. This means that it cannot be very large, or protrude out of the water more than a few inches. Typical ocean uses of GPS, such as the buoys described in [BROW93] and [LEU93] are designed to place the antenna well out of the water. With an antenna that is located on or just above the surface of the ocean, wave-wash will obscure the antenna. The frequency and the length of time that the wave wash would cover the antenna, as well as the depth of the water, will effect the ability of the receiver to obtain a GPS position fix. Chapter IV explains the tests that were run to determine the extent of the degradation in the ability to use GPS navigation under these circumstances.

C. SUBMERGED NAVIGATION

As previously stated, the mission of the AUV under study [HEAL94] and the SANS is to provide precise position information relating to a submerged object. In the interim system, when an object is found during the mapping phase, the AUV must surface. In order to determine the distance and direction from the submerged object to the surface, where a GPS position fix can be obtained, the depth change, heading or direction of travel, and the climb angle must be known. With this information the submerged location can be extrapolated from the surfaced location. Figure 1 shows an arbitrary path of a surfacing AUV. The average climb angle is represented by θ, Z is the depth change and H is the horizontal distance traveled. The horizontal distance is calculated using Equation 1. In

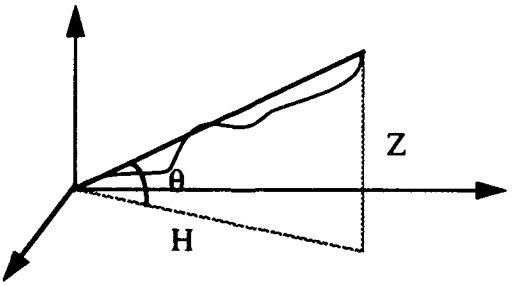


Figure 1. Computation of Horizontal Distance Travelled from [KWAK93]

order to improve the accuracy of the computation, these measurements are recorded at small intervals which SANS sums to obtain the final resulting distance.

$$\Delta H = \frac{\Delta Z}{\tan\theta} \qquad \text{Eq 1}$$

These measurements are obtained from various sensors contained in the SANS. The depth change is measured by a depth transducer and the heading is measured by a flux-gate compass. Two approaches to measuring the climb angle of the AUV were described in [STEV93]. The first approach used a miniature gyroscope. The orientation of the AUV when a submerged object is located was assumed to be near level. Since the gyro measures only relative position, the climb angle would be determined relative to that near level position. The climb angle is computed based on the change in orientation of the AUV. The second approach is to determine the climb angle using an accelerometer (inclinometer). This approach and the inherent errors in its use are described in [MCGH93]. Specifically, an accelerometer measures the total acceleration of an object. The output of a longitudinally oriented accelerometer is shown in Equation 2, where a is the longitudinal acceleration of the AUV, g is gravitational acceleration, and θ is the climb angle.

$$sensed-acceleration = a + g\sin(\theta) \qquad \text{Eq 2}$$

When the AUV begins to surface, it will have to accelerate. However, if the AUV has constant velocity, then the longitudinal acceleration is zero, $a = 0$, and the equation is simply

$$gravity-component = g\sin(\theta) \qquad \text{Eq 3}$$

so

$$\sin(\theta) = gravity-component/g \qquad \text{Eq 4}$$

The equation normally used for determining the pitch attitude of a static object with an inclinometer is Equation 4. But because the accelerometer is on a moving platform, this must be modified to Equation 5.

$$\sin(apparent-\theta) = sensed-acceleration/g \qquad \text{Eq 5}$$

From Equation 2 and Equation 5, we get Equation 6, where a/g represents the error between the sin of the actual climb angle and the sin of the apparent climb angle sensed by the accelerometer. That is,

$$\sin(apparent - \theta) = \sin(\theta) + a/g \qquad \text{Eq 6}$$

So,

$$a/g = \sin(apparent - \theta) - \sin(\theta) \qquad \text{Eq 7}$$

From Equation 1 and Equation 7, the horizontal error introduced by the error in the climb angle is calculated as:

$$horizontal - error = \Delta Z\left(\frac{1}{\tan(\theta)} - \frac{1}{\tan(apparent - \theta)}\right) \qquad \text{Eq 8}$$

Therefore, the apparent climb angle of the AUV can be calculated based on information from the accelerometer, and with the depth change from the depth transducer, the horizontal distance can be computed, again using Equation 1. Further explanation of the calculations required are presented in Chapter V, Section E. The error in this calculation comes from the effects of longitudinal acceleration. Also, in this system configuration, the accelerometer would be mounted along the body axis of the AUV. If the velocity vector of the AUV deviates from the longitudinal body axis, this would add an additional error. By performing bounding calculations, McGhee concluded in [MCGH93], that the error generated by an accelerometer used for determining climb angle is acceptable, in part because of the motion limits of the AUV. Additionally, as the AUV reaches the surface, it will decelerate. This deceleration will tend to cancel out the errors created from the acceleration. These effects will be treated quantitatively for typical surfacing maneuvers in Chapter V.

D. NAVIGATION SIMULATION

A computer simulation of an AUV with the SANS has been created in order to simplify the process of determining the errors created by the sensors. This simulation is written in CLOS (Common LISP Object System), an object-oriented programming language. This allows the AUV and the SANS to be represented as objects and classes that

are put together to create an entire system. The simulation is designed as an AUV which has the physical properties of a rigid body and a separate SANS which is also a rigid body. The SANS has a link to the AUV to allow them to move as one unit. This unit can move as required to simulate the actual mission. Using forward kinematics and dynamics based on Newton-Euler equations [FU87], the final Cartesian coordinates of the system can be determined.

CLOS also permits each class to have local and shared slots with accessible slot values [KOSC90]. Within the SANS simulation, each sensor has a slot containing the measurements that would be obtained by that sensor in the actual navigation system. This simulation allows for making various changes to the parameters of the AUV mission, or sampling a number of missions. By running a new simulation, an estimate of the resulting error can quickly and easily be determined. For example, the simulation can be run using any climb angle or any depth desired to determine how these changes effect the total error. The equations for determining the errors in the simulation are the same ones explained above and in Chapter V, Section E. A graphical representation of the AUV and SANS was included in the simulation as a way of visually demonstrating the approximate error of the system. This simulation could be tailored to different types of AUVs, using different types of sensors as well. The simulation program is explained in detail in Chapter VI.

E. SUMMARY

Navigation system design for an AUV is greatly influenced by its mission. Therefore the mission requirements have been explained. GPS navigation has some unique problems to overcome in order to allow the AUV to remain undetected when surfaced. This research continues the evaluation of the interim system described in [STEV93], determining how the SANS will perform when used in the ocean. This study includes analysis of data collected from experimental testing in the lab tc determine the performance of the GPS antenna and receiver with limited exposure time. It also includes experimental testing in the lab and on the ocean to determine the effect on GPS navigation of the wave wash over the antenna.

The use of an accelerometer, instead of a gyroscope, to determine the climb angle of the AUV is explored in this research. The accelerometer provides the sensed acceleration of the AUV, which is composed of the AUVs longitudinal acceleration plus the gravitational acceleration multiplied by the sin of the climb angle as seen in Equation 2. Therefore, with the additional information provided by the depth transducer and the flux-gate compass, the climb angle and heading can be calculated. A graphical computer simulation has been developed to simplify the process of determining the climb angle based on the equations described, and the error of this system under various mission parameters using various system configurations has been examined. Simulation results can be used to determine the accuracy requirements of the components in order to meet the overall system accuracy goal. Any motion restrictions or requirements that may be imposed on the AUV can also be included in the simulation.

IV. SYSTEM DESIGN

A. INTRODUCTION

The Small Autonomous Underwater Vehicle Navigation System (SANS) design studied in this research is essentially the same as the interim system described in [STEV93] and [KWAK93] with the exception of a new version of the core module, a new, smaller A to D converter, and an accelerometer rather than gyroscopes to determine the climb angle of the AUV while surfacing. Only computer simulation results of the change to an accelerometer are given. No experimental testing of the complete system has been performed with the new equipment, although considerable evaluation of components has been conducted as part of the work of this thesis.

The hardware for this interim system was chosen to meet the requirements of size, weight, power and cost. There are many alternative choices for commercially available hardware, and there are inertial navigation systems which would meet or exceed the requirements for this project. For example, Systron Donner Corporation has a small high performance inertial measurement unit (IMU) [GYRO92] that is expected to meet the requirements of this mission. Additionally, its accuracy is sufficient to lenghten the time required between updates of the sensors to allow the AUV to locate multiple targets before surfacing to obtain a GPS fix. The baseline system as described in [MCGH92] would include the hardware needed to accomplish this. However, the cost of those sensors prohibited them from being included in this testing. This chapter describes the hardware used in this modified interim system. Technical specifications of each device are included in Appendix A.

The software design of the SANS has not been modified in any way since it was described in [STEV93]. It is an object-oriented design in Ada, using assembly language for low level, high frequency operations. The software is made up of three primary operations related to the mission. These operations are monitoring the AUV, navigation data-logging, and GPS data-logging. Each of these operations consists of software objects, and operations performed within that object that are related to individual sensor data. In order

to ensure that the software could be adapted as the SANS evolves, the software was designed with code reuse as an important feature. A high level description of the software is included in this chapter.

B. HARDWARE DESCRIPTION

The hardware design chosen to accomplish the mission requirements described in Chapter III are shown in Figure 2. The total volume of the equipment must not exceed 120 cubic inches, therefore size was a primary concern in choosing hardware. Recent advances in miniaturization have made the task of finding adequate equipment in small sizes much easier, but miniaturization adds to the cost of the equipment.

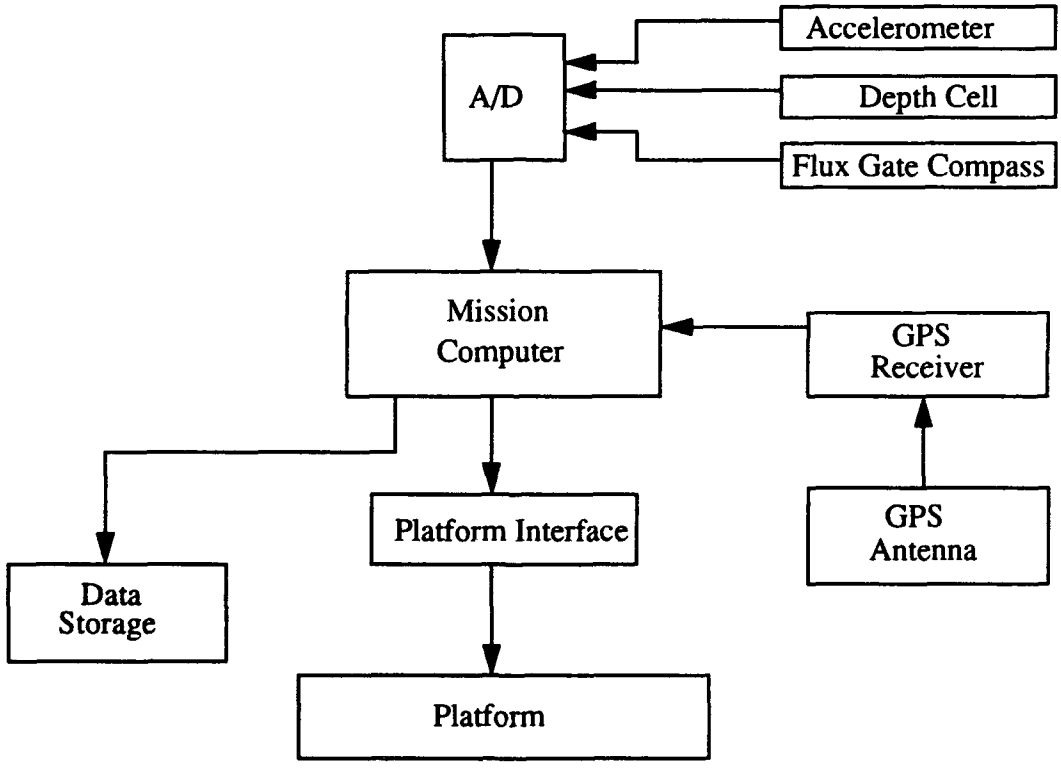


Figure 2. Hardware Block Diagram for Interim SANS after [KWAK93]

1. Mission Computer

Dovatron E.S.P. 8680 (Extra Small Package) core module, shown in Figure 3, was designed specifically for the developer who has space and power limitations which require

off-the-shelf PC-compatible solutions. The E.S.P. provides small size, modularity, PC-compatibility, low power consumption, and availability of many off-the-shelf modules. The core module contains a graphics controller, serial interface, memory, keyboard controller and ROM-based software. It can be used with any IBM-XT compatible keyboard and monochrome or color CGA monitor. The core module contains a 14 MHz Chips and Technologies F8680 PC/CHIP, 512 kilobytes or 1 megabyte of dynamic RAM, ROM-based software, a PCMCIA connector which can have up to 32 MB of memory, and the necessary support chips. The core module memory can be expanded using additional modules that plug into the backplane. [ESP93-1] The core module is 1.7 x 5.2 x 4.7 inches (41.54 cubic inches), and all modules are designed to conform to that standard E.S.P. dimension.

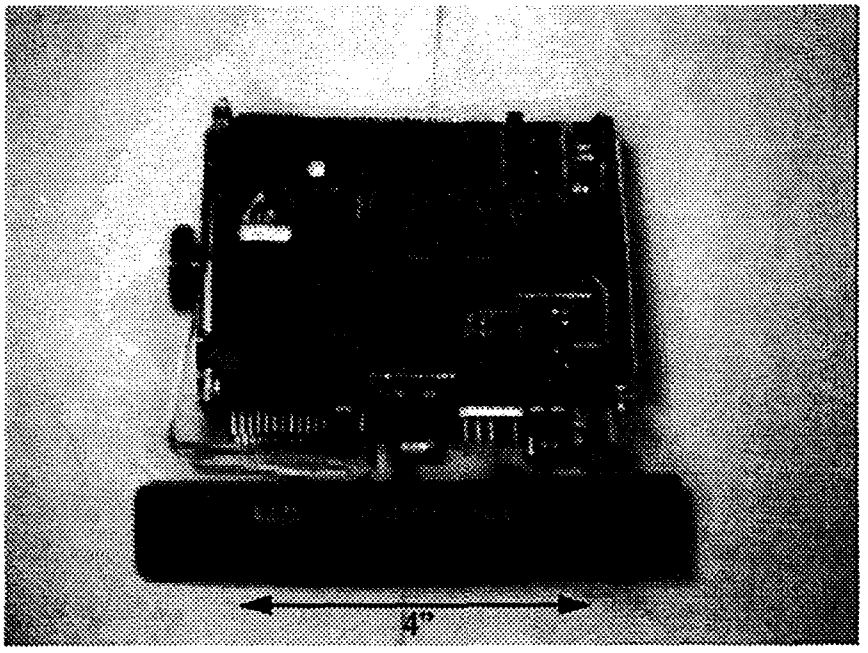


Figure 3. E.S.P 8680 Core Module

In order to conserve power, there are four different power modes. "Run" mode is the normal operating mode which consumes the most power but offers full PC/CHIP performance. The "drowsy" mode uses PC/CHIP hardware to insert delay cycles between instructions. This is typically used when performance is not important, but low-level

processing is required. "Sleep" mode stops all of the clocks to the PC/CHIP, and they are not restarted until an interrupt or DMA request occurs. In the "suspend" mode, the real-time clock inside the chip is powered, and the external memory may be powered. This allows the application to be suspended, and it can be resumed at any time. An input to the PC/CHIP is used to control the entry into and exit from suspend mode. The software will bring the system to an orderly shutdown and when returning to normal operations will restore the clocks and power to the module, with all internal registers and memory preserved. System power can be reduced to microwatts while in suspend mode. [ESP93-1]

The E.S.P. 8680 comes with a Flash EPROM (Erasable Programmable Read Only Memory), which is an electrically-erasable ROM that can be reprogrammed. Using Flash memory technology, this add-on card emulates an ordinary magnetic Hard Disk. The disk portion of the ROM can be read, written and edited without removing the ROM or using additional devices. It is mounted in a standard 32-pin PLCC socket. The E.S.P. Flash Disk is a memory card which is non-volatile, programmable and low power, that appears just like a hard disk drive to the operating system and can be used with any program that runs under DOS. No device drivers or other software are required. [ESP93-3] The Flash EPROM is available in capacities from 1 to 16 MB, and is 1.7 x 5.2 inches, the standard E.S.P. form factor.

The E.S.P. 8680 has a PC Memory Card International Association (PCMCIA) connector and electrical interface for memory cards. The interface supports ROM, RAM and flash memory cards, and can provide up to 32 MB of memory. [ESP93-1]

2. A to D Converter

The Dovatron E.S.P. Analog to Digital Module, as seen in Figure 4, is based on an existing ISA plug-in card. It provides an interface for data acquisition, control, monitoring, and parameter measurement. This module provides for measurement of up to 16 single-ended channels (in sequence mode, the sequencer address starts at 0, and progresses until the address count is equal to the count selected in the program, so only 15 channels are

usable), or 8 channels in differential mode, and includes many of the features normally found on full sized PC peripheral A to D cards. The module uses an Intel 82C54 programmable interval timer which provides three independent 16-bit counters, each with six programmable counter modes and either binary or BCD counting. Again this E.S.P. converter is in the standard size of 1.7 x 5.2 x 0.7 inches. [ESP93-2]

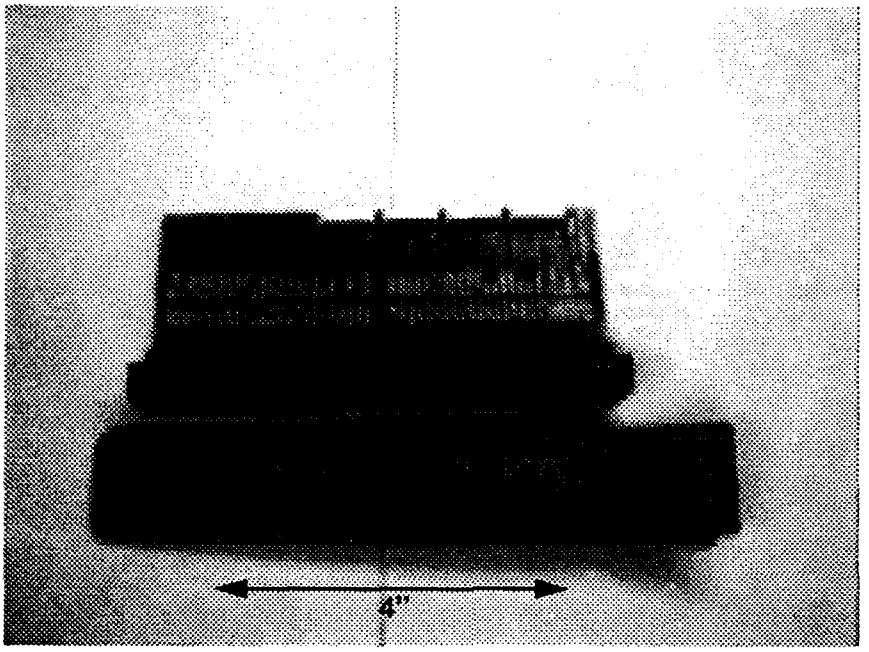


Figure 4. E.S.P. A to D Converter

3. GPS Receiver

The Motorola PVT6 receiver, shown in Figure 5, is capable of tracking six satellites simultaneously and of performing real-time differential processing. It can provide better than 25 meter position accuracy and 0.1 meter/second velocity accuracy without selective availability (S/A) on, without differential processing. Differential processing improves the accuracy to one to five meters. It can be powered with unregulated 12 Vdc or regulated 5 Vdc. Keep-alive random access memory (RAM) allows the receiver to retain satellite ephemeris data and real-time-clock (RTC) information which require an external 12 V or

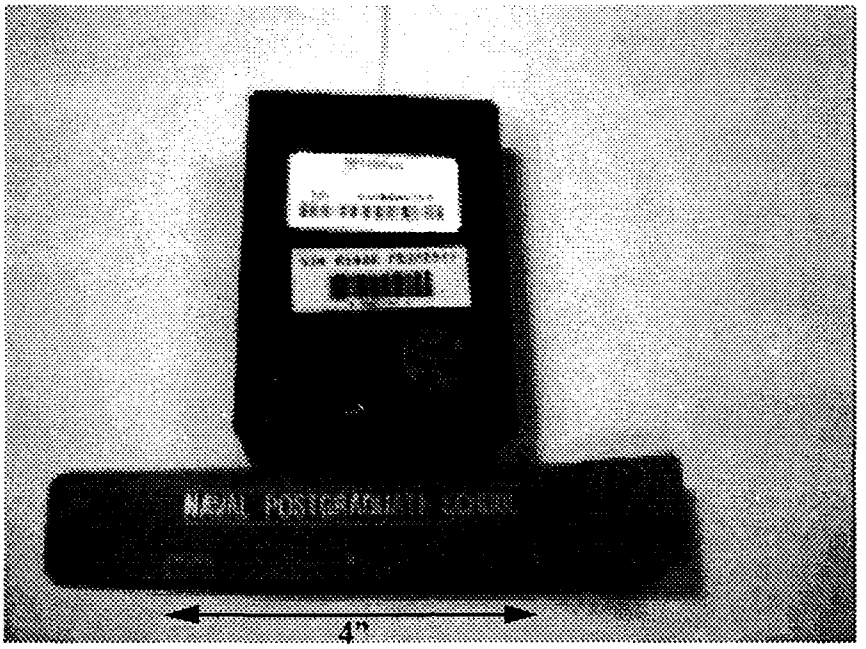


Figure 5. Motorola PVT-6 GPS Receiver

5 V battery. The receiver is 3.94 x 2.76 x 0.65 inches (7.06 cubic inches), weighs 4.5 oz., and uses 1.3 W of 5 Vdc input or 1.8 W of 12 Vdc.

There are four different power-up states, cold-2, cold-1, warm and hot as shown in Table 2, from Motorola's GPS Receiver Technical Manual. Cold-2 is the state the receiver

POWER-UP	INITIAL ERROR			AGE		TTFF	
STATE	POS	VEL	TIME	ALMANAC	EPHEMERIS	Typical	90%
Hot	100 km	75 m/sec	3 min	1 month	<= 4 hrs	21 sec	30 sec
Warm	100 km	75 m/sec	3 min	1 month	U/A	53 sec	66 sec
Cold - 1	N/A	N/A	U/A	1 month	U/A	7.9 min	15.0 min
Cold - 2	N/A	N/A	N/A	U/A	U/A	13.0 min	25.0 min

Table 2. TTFF Information from [MOTO93-1]

is in the first time it is turned on, or any time the receiver is without power for more than a month. The receiver will not have the current date and time and will have to search for all available satellites. After it finds the first satellite, the date and time are corrected and it will continue to acquire other satellites. Cold-1 state occurs when the receiver has lost power,

so the RTC and the RAM are erased, but the almanac is less than a month old. A cold start is quite slow, but if the receiver maintains its battery power, a cold start is required only the first time it is used. Once the receiver is tracking three or more satellites, position computation is done automatically. The receiver stores the almanac and the ephemeris data as well as the position of the receiver and the time in RAM, making it much faster to acquire satellites from a warm or hot start. A warm start is when the position and time are known, but the ephemeris data is too old to be useful, about four hours. A warm start has a Time-To-First-Fix (TTFF) of about one minute. Finally, a hot start provides the fastest TTFF, usually under 30 seconds. It is able to use the ephemeris data stored in RAM, along with the position, time and almanac to find satellites. The mission of the AUV will allow the SANS to always remain in a hot state after launch. Reacquisition time is the time it takes for the receiver to reacquire the satellite signals that are lost when the signals are obscured. As seen in Table 3, from Motorola's GPS Receiver Technical Manual, this time is a function of the time that the signal is obscured. Chapter V provides TTFF and reacquisition

TIME OBSCURED	REACQUISITION TIME (Sec) (Typical)
15 sec	< 2.5
30 sec	< 3.5
45 sec	< 3.5
60 sec	< 3.6

Table 3. Reacquisition Time from [MOTO93-1]

time obtained in experiments conducted as part of the research of this thesis.

The PVT6 is capable of providing autonomous position, velocity, and time information over a serial RS-232 port. There are three different format types available for the data that can be selected by the user. These are the Motorola Binary Format, National Marine Electronics Association Standard (NMEA) Format 0183, and the LORAN Emulation Format. The Motorola Binary Format was chosen for use in the SANS because it provides both satellite range information and position format messages. The following is a complete list of output message types available:

- Position/Channel Status
- Satellite Range Data Output
- Pseudorange Correction Output
- Ephemeris Data Output
- Visible Satellite Status
- DOP Table Status
- Almanac Status
- Almanac Data Output
- Leap Second Pending

Each output message can be provided one-time or continuously at a selected rate, and stored for post-processing. The requested output options are stored in nonvolatile memory, and will remain in effect when the receiver is powered up again. This allows the user to select all desired options prior to a mission, and the receiver will remember those options until they are manually changed. [MOTO93-1]

4. Compass

The KVH C100 Multi-Purpose Digital Compass, shown in Figure 6, is a flux gate compass which provides three optional outputs. A 4 digit Binary Coded Decimal (BCD) serial stream formatted digital output is available, and both a linear and sine/cosine voltage signal analog output are available. Because the digital output would require an additional serial port connection to the E.S.P. 8680, the analog output is used. The analog output signal is processed through the A to D converter. The linear voltage output is proportional to the compass heading from 0.1 volts (000 degrees) through 1.9 volts (360 degrees). This voltage is converted to digital data by software triggered A to D conversion mode through the A to D converter. The analog signal from the compass is connected to the converter through one of 8 multiplexed input channels. This input channel is selectable at run-time

through software. The compass is 1.8 x 4.5 x 1.1 inches (8.91 cubic inches), weighs 2.0 ounces, draws 0.1 W at 5 V dc current and is accurate to $\pm 0.5^{o}$. [KVH91]

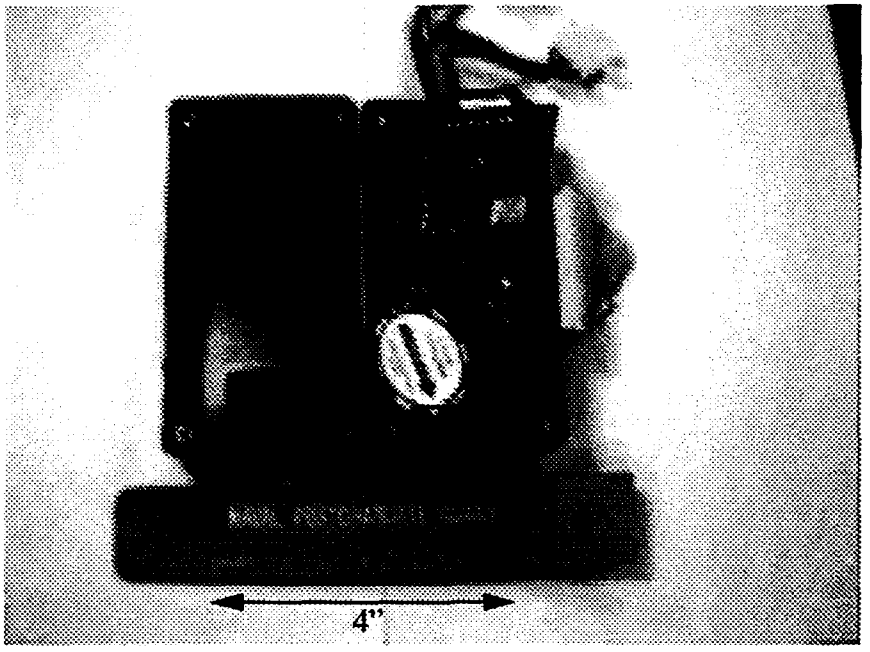


Figure 6. KVH C100 Multi-Purpose Digital Compass

However, this accuracy only applies to a platform with a tilt angle of $\pm 16^{o}$ if the compass is mounted to the platform, or $\pm 45^{o}$ if the compass is mounted on a gimbal which suspends the compass in the horizontal plane. This would be acceptable for many AUVs, but not all. For example, a human diver or a dolphin (which constitute biological AUVs from the perspective of this research) would tend to dive and surface at angles greater than $\pm 45^{o}$. Additionally, a gimballed system is susceptible to dynamic vibrations, shock and acceleration forces of the platform [HELL92]. These factors would not be critical for a mechanical AUV that can only move and accelerate slowly, and that is restricted to very shallow climb angles. But, another type of sensor must be used to determine the heading for other types of AUVs.

A solution to this problem is a strapdown sensor which provides three axes of magnetic measurement in a single package. The sensor is solid-state with the axes aligned

to the mechanical datum of the sensor case. Three-axis strapdown magnetometers provide better accuracy than a gimballed compass, and do not require restrictions on the tilt angle of the AUV. [HELL92] An example of this technology is the TCM1 Electronic Compass Sensor Module from Precision Navigation. It contains a tri-axial magnetometer and a bi-axial electrolytic inclinometer in a 2.5 x 2 x 1.1 inch package [PREC94]. Another advantage of this sensor is that it provides compass heading and pitch and roll readings, in one sensor, which reduces the number of serial interfaces required in a system.

5. Depth Transducer

The Omega Inc. PX176-100S5V Depth Transducer, as seen in Figure 7, is a small cylinder 2 inches long and 1.5 inches in diameter (3.53 cubic inches), weighs approximately 5 ounces and has an operating range of 0 to 100 pounds per square inch of static pressure (PSIS). This equates to 6.7 standard atmospheres or 217 feet (67 meters) depth of sea water. Analog output is one to six volts dc. A to D conversion is performed by repetitive software triggered single A to D conversions in the same manner as the compass signal with the analog input channel also selectable through software. [PARK80]

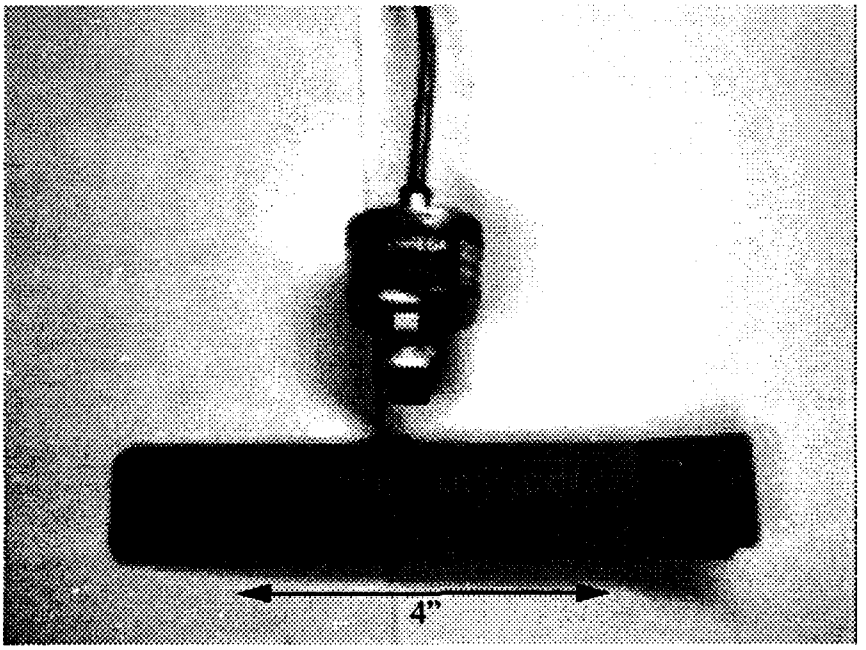


Figure 7. Omega Depth Transducer

6. Accelerometer

The Humphrey LA67-0108-1 Linear Accelerometer, seen below in Figure 8, is available in a measurement range of +0.5 to +1.5 G, is packaged as a small cylinder 1.8 inches long and 1 inch in diameter (0.45 cubic inches), weighs 4 ounces, and provides accuracy to ±5% of full scale with light vibration, applicable between 10 and 90% of full scale. The accelerometer measures acceleration using solid state silicon resistors which change electrical resistance in proportion to the applied mechanical stress. While this

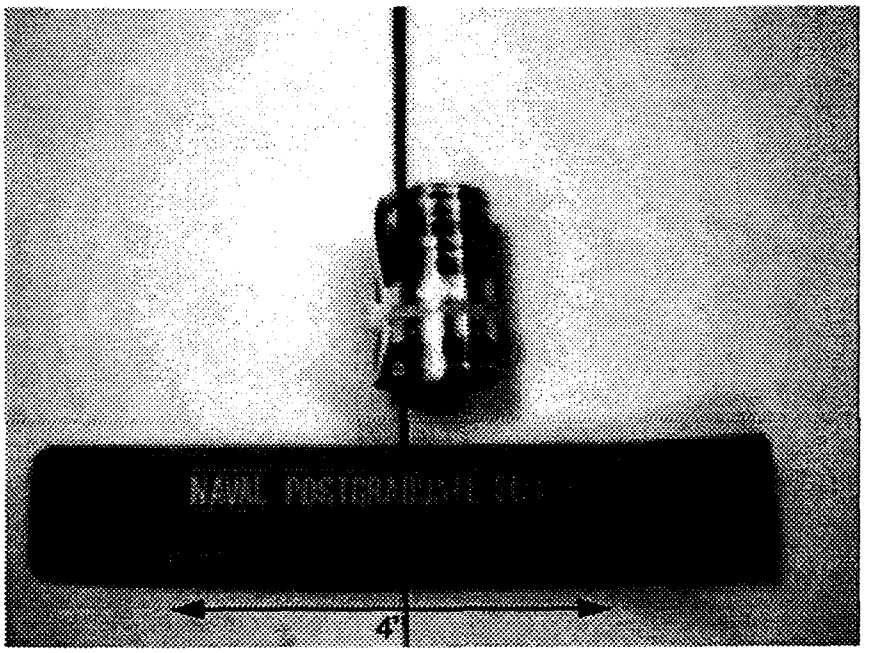


Figure 8. Humphrey Accelerometer

particular accelerometer is marginally acceptable with regard to accuracy, recent technology provides much more precise measurement in even small packages [GYRO92].

C. SOFTWARE DESCRIPTION

As developed and described in [STEV93] and [KWAK93], the SANS software design has three major operations. These operations are:

- Monitoring the AUV for a position fix request

• Navigation data-logging for dead reckoning (DR) navigation (post-processed to determine the ascent vector)

• GPS data-logging for post-processed positional information

AUV monitoring must be performed continuously during the mapping phase in anticipation of a request by the AUV to determine the position of an underwater object of interest. Monitoring of the AUV will be done using RS-232 serial line communication. GPS and dead reckoning navigation will remain inactive until the AUV requests a position fix, then GPS will be powered. This allows the GPS receiver to be unpowered as much as possible to conserve battery power.

The GPS receiver will be initialized with the proper receiver modes required for the mission. The Motorola Binary Format mode will be used, and the receiver will transmit both position/status and satellite range format messages at a one second rate. The computer will be configured to receive RS-232 serial communications with connection parameters specified from an input data file at run-time. The GPS output messages will be written to non-volatile memory on the host machine prior to real-time processing. The satellite range data will be stored for post-processing only. The position/status messages will be processed for navigation updates, and the position data will be used to determine the AUV's current position using SPS and real-time differential processing.

Submerged navigation in the SANS is accomplished by determining the direction and horizontal distance from the submerged object to the surfaced location using the depth change, heading and climb angle, as described in Chapter III, Section C. The outputs will be pitch attitude (or climb angle)and heading in degrees, and depth in meters, with a precision adequate to satisfy system accuracy requirements as stated in Chapter III, Section A.

The interim SANS software programming language is Meridian Ada version 4.1.1. Assembly language is used for low level, high frequency operations in order to improve efficiency. An object-oriented design was chosen for the SANS because it allows for a simpler management of this complex system. Further explanation and software testing is described in [STEV93].

D. SUMMARY

The interim SANS design described in [STEV93] is the basis for the system described in this report. The use of gyroscopes in that system would require a limitation on the maximum climb and dive angle of the AUV to prevent gyro tumbling. Therefore, the research of this thesis explores replacing the gryos with a linear accelerometer to determine the climb angle of the AUV. Also, a recommendation for changing the compass to a three-axis magnetometer is made, because the compass would also require restricting the climb angle of the AUV. A new version of the E.S.P. 8680 core module and a new, smaller E.S.P. A to D converter replace those hardware pieces from the former system. The hardware for the SANS was chosen with the mission requirements in mind. Although there are many possible choices of off-the-shelf hardware for each part of the system shown in Figure 2, they all must be able to provide the accuracy required, as well as being able to meet the size, weight, power, and cost constraints of the proposed system. Therefore, possible systems which may provide better accuracy were not studied due to the increased cost. As further advances in miniaturization are made, the cost of this hardware will be reduced, making it easier to meet the baseline SANS requirements explained earlier.

The software design described in [STEV93] is an object-oriented design programmed in Meridian Ada and assembly language. It is a modular system designed to make code reuse easy as the SANS hardware and software systems evolve. The software is broken down into the three logical primary operations of the mission, monitoring the AUV, navigation data-logging and GPS data-logging. Changes can be made to the objects and operations in the software as may be required for future research.

V. DEMONSTRATION SYSTEM TESTING

A. INTRODUCTION

The technical specifications of the hardware for the SANS, discussed earlier in Chapter IV and listed in Appendix A, were provided by the manufacturers of the equipment. This chapter will present the methods and results of experimental testing performed to determine the accuracy and speed of acquisition of the GPS receiver under various circumstances. Static testing was performed in an attempt to confirm Motorola's specifications given in [MOTO93-1]. Additional static testing was performed to obtain data related to the time required for tracking three satellites after being powered off. This information is used to determine the time required before differential processing could be used to determine a more accurate position fix.

In order to determine the extent of degradation of the GPS signal caused by standing water covering the antenna, a series of bench tests were performed. This was designed to relate to the effects of the wave wash under normal conditions at sea. Finally, an actual sea test was done to determine the effects of normal wave-wash over the GPS antenna, and executing frequent dives, while recording position and range data.

B. STATIC TEST RESULTS

The acquisition time and accuracy of the GPS receiver are the limiting factor in the overall accuracy of the SANS. Experimental static testing was performed to confirm the results presented in Chapter IV. To determine the acquisition time, or Time-To-First-Fix (TTFF) of the receiver in both hot and warm power-up states, samples were taken with the receiver off from 90 seconds to 6 hours. A GPS antenna was mounted on the roof of Spanagel Hall at the Naval Postgraduate School, at a known, surveyed location, nearly free of any signal obstructions. The GPS receiver was connected to this antenna, a PC, and a relay box. This relay box is used to interrupt the 12 volt main power supply to the receiver for a specified time. The receiver remained connected to the 5 volt battery or "Keep Alive"

power the entire time. This allowed the receiver to retain the real-time clock and the last known coordinates stored in RAM.

The relay box used in this testing was controlled by a general serial I/O handler called GEORGE, written and developed by Dr. James Clynch. GEORGE is capable of providing various functions that act as a terminal emulator through a PC parallel port [CLYN92-2]. Control programs in GEORGE were written to send signals at specified times to the relay box. For example, in order to determine the TTFF of the GPS receiver after 90 seconds off, GEORGE would signal the relay box to turn on for 2 minutes, then turn off for 90 seconds, and loop. The receiver was set up to write position and range data to a file every second. This permitted recording of the acquisition time and the number of satellites tracked after power was returned to the receiver. The results of these tests are shown in Appendix C, and are summarized below in Table 4. The TTFF of the receiver is

RECEIVER TIME UNPOWERED	ACQUISITION TIME FOR FIRST SATELLITE (rms) SECONDS	ACQUISITION TIME FOR THREE SATELLITES (rms) SECONDS	FIRST FIX ERROR (rms) METERS	NUMBER OF SAMPLES
90 SECONDS OFF	23.18	23.89	34.69	63
10 MINUTES OFF	22.18	24.01	44.78	35
30 MINUTES OFF	23.97	27.05	32.95	35
1 HOUR OFF	28.79	35.83	33.30	45
3 HOURS OFF	24.46	37.76	41.80	31
6 HOURS OFF	44.33	46.21	30.91	8

Table 4. GPS Static Test Results Summary of Time-To-First-Fix

fairly consistent through three hours off, which is the expected result since the ephemeris data is still current enough to be useful in acquiring satellites. The first fix position error is also fairly consistent, and consistent with the SPS level of performance.

Observing three satellites will provide two-dimensional position and velocity of the AUV. Therefore the acquisition time required to track three satellites, and the accuracy of the position data when tracking three satellites, are also extremely important information in assessing the overall accuracy of the receiver. The time required to obtain three satellites increases with the increase in power-off time. The mission requirements are for the AUV to limit antenna exposure time to 30 seconds, with intervals between exposures of at least several minutes. The data in Table 4 indicates that these requirements can be met for update intervals of up to 30 minutes. The first fix from the testing showed that the accuracy of the receiver is well below the 100 m accuracy available using SPS. The receiver accuracy will not degrade the overall system accuracy. And with differential post-processing, the position error should be reduced to an accuracy of 2 to 4 meters.

C. GPS BENCH TEST RESULTS

As stated in Chapter III, during both the transit phase and the mapping phase of the mission, the AUV needs to be as difficult to detect as possible. Thus, the GPS antenna mounted on the top of the AUV must not protrude out of the water more than a few inches. With so little clearance above the surface, normal ocean waves would repeatedly cover the antenna for short periods of time. This wave-wash would typically last for only a few seconds at a time, but would occur continuously throughout the mission. One of the primary goals of this thesis was to determine whether the effect of the wave-wash would degrade the signal beyond use. In order to determine if the mission requirements could be met, it would have to be known whether a shallow layer of sea water covering the antenna would completely obstruct the GPS signal. As a preliminary step in determining this effect, a bench test was designed. A GPS antenna was mounted on a wooden platform, designed specifically for this test, at the height of the side supports. The platform side supports hold a plastic container, which was used because the plastic does not interfere with the GPS signal. The bottom of the container sat flush with the antenna. This test equipment is shown in Figure 9. The bench test was performed on the roof of Spanagel Hall, where there are no

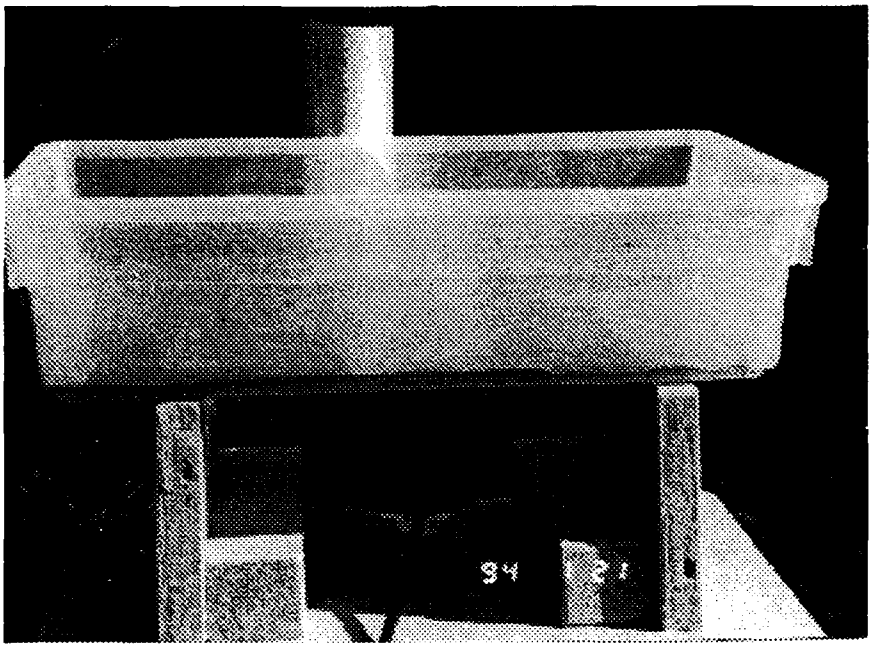


Figure 9. Bench Test Equipment

signal obstructions. The antenna was connected to the Motorola GPS receiver with 60 feet of RG-213 cable, and a laptop computer was used to control the receiver. In two of the test cases, there were only five satellites visible at the beginning of the test. The other two test cases started with the receiver tracking six satellites. With the receiver running, sea water was poured into the plastic container every two minutes. Each pour added a depth of 0.5 mm of water over the antenna. The effect on the number of satellites tracked was recorded, along with the water depth, and which satellites were being tracked. A graph of the data is shown in Figure 10. This shows that a depth of between 3 and 5 mm of water can cover the

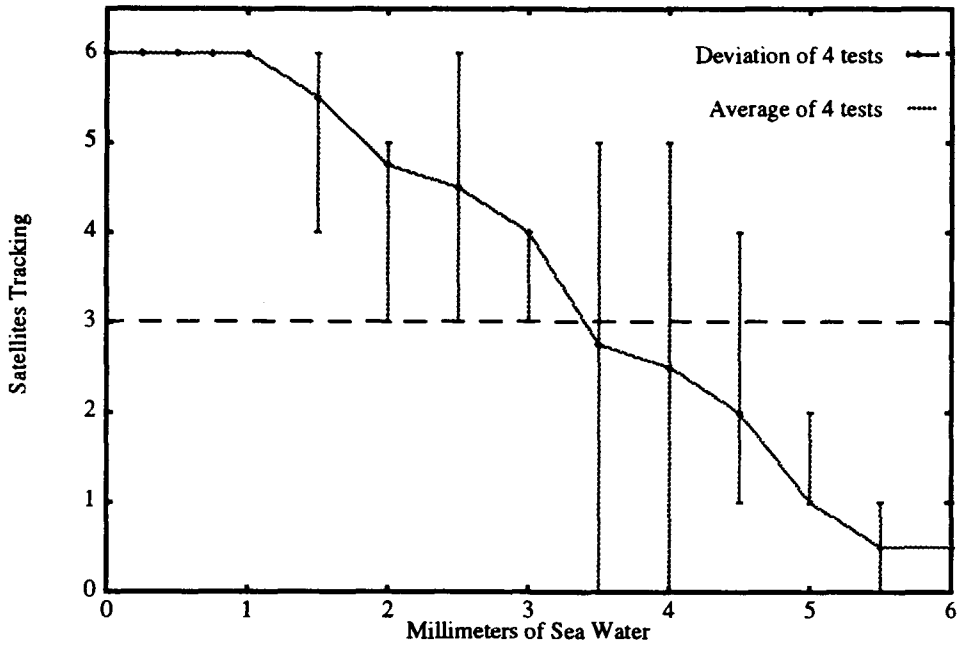


Figure 10. Bench Test Results

antenna, and the receiver can still track at least three satellites for use in differential post-processing. In this test, the water was left covering the antenna continuously, until all GPS signals were lost. Because actual waves would wash over the antenna very quickly, reacquisition time for additional satellites after the wave recedes, would be very short, typically less than 3 seconds.

Analysis of an elevation graph of the GPS satellites from the day of the test show that the satellites that the receiver was able to continue tracking, even with 4 - 6 mm of water covering the antenna, for those satellites that were at the highest elevation. The lower the elevation of the satellite during the test, the faster the signal was lost with water over the antenna. In two different tests, the receiver was able to continue to track one satellite with 11 mm of water covering the antenna. In both cases, that satellite was almost directly overhead the antenna during the test. The reason for this is due to the actual path of the signal underwater. As shown in Figure 11, when the antenna is covered by 5 mm of water,

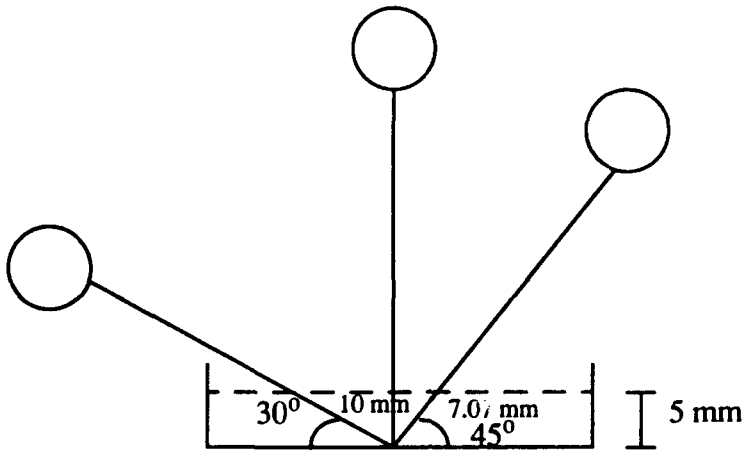


Figure 11. Diagram of Satellite Elevations with Effective Depth of Water

if a satellite is overhead, the signal path travels 5 mm through the water. If the satellite is at 45° the path through the water is 7.07 mm. If the satellite is at 30°, the path travelled is 10 mm. The effective depth of the water is the least when the satellite is directly overhead at 90°.

The antenna that was used for this test was the antenna that was included in the Motorola GPS evaluation kit, along with the receiver. This antenna module is a low-profile antenna with a protective housing that is 4.01 inches in diameter and 0.89 inches thick, and is the black antenna shown in Figure 12. There are various shapes and sizes of antennas that could be used. For testing purposes a relatively flat antenna was required to allow the water to sit over the antenna. However, for the baseline SANS, a fin-shaped antenna would probably provide better GPS accuracy and shorter acquisition times since more of the

antenna would be out of the water. A smaller, flat antenna could also be used for minimum detection. A sample of these antennas is shown in Figure 12.

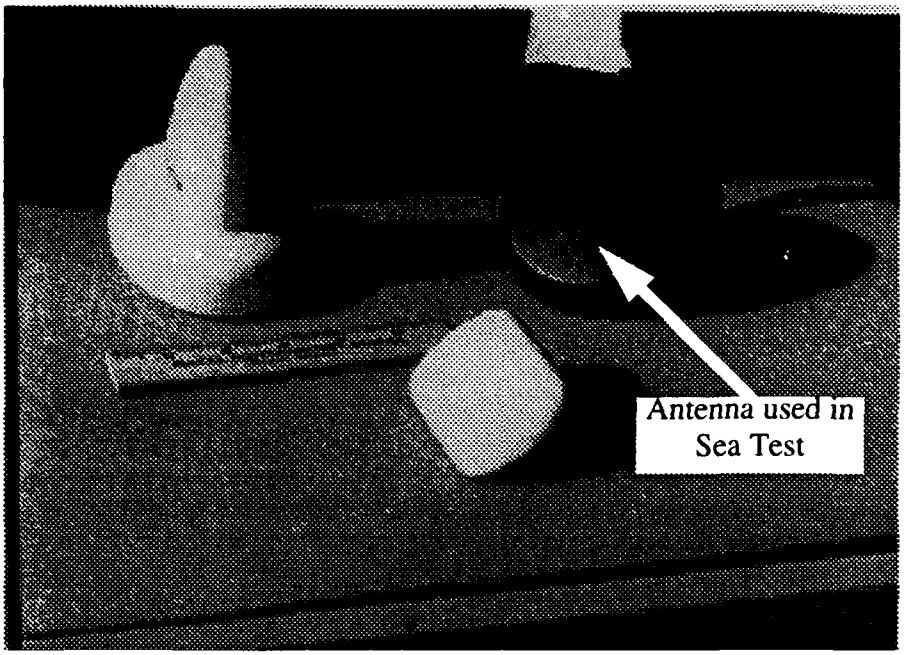


Figure 12. Sample GPS Antennas

D. SEA TEST RESULTS

After establishing that the proposed mission requirements could be met based on the static tests and the bench tests, a sea test was developed. The purpose of the sea test was to further determine the ability of the GPS receiver to track satellites and obtain position and range data while the antenna was in the ocean. The first step of this test was to build a test vehicle that could be used as a limited simulation of an AUV, with the ability to float on the surface with the antenna just above the water, and with the ability to dive and surface as needed. A wooden platform was built, shown in Figure 13, which could be towed behind a boat, with a piece of metal attached to the rear that could be operated like an airplane

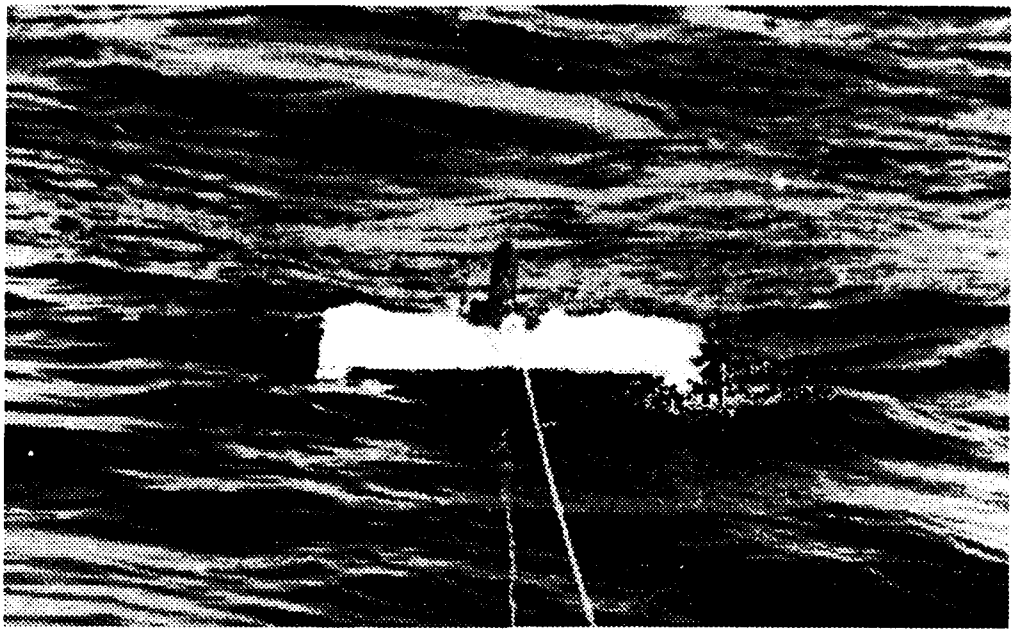


Figure 13. Sea Test Platform

elevator by pulling on an attached line. This photo shows a preliminary test of the platform, with a wooden replica of an antenna mounted in the center. The sea test was performed by towing the platform behind a sailboat in Monterey Bay. This allowed a fairly constant speed in the water to properly control the platform, with no propeller interfering with the lines used to control the depth of the platform.

The actual sea test was performed on a cool, sunny afternoon with NW winds at 4 knots and 6 ft swells. The flat Motorola antenna that came with the GPS receiver was used for the actual test to evaluate the worst case, because the water can sit on top of it. The antenna was connected to the receiver on the boat, with 60 feet of RG-213 coaxial cable. A laptop computer was used to view and record the position and range data every second throughout the test. The test was also recorded on video tape to allow for later re-evaluation of the test along with the recorded data.

Initially, the equipment was connected and turned on with the platform in the boat, and the receiver was able to track six satellites. For one hour and six minutes, the platform and antenna were towed behind the sailboat. The sea test maneuvers that were performed

were in the range of ocean wave occurrences that would be likely during the transit phase of the mission. The maneuvers that were performed and recorded for this test included long, deep dives to depths of 20 feet or more, such as that seen in Figure 14. During the long dives

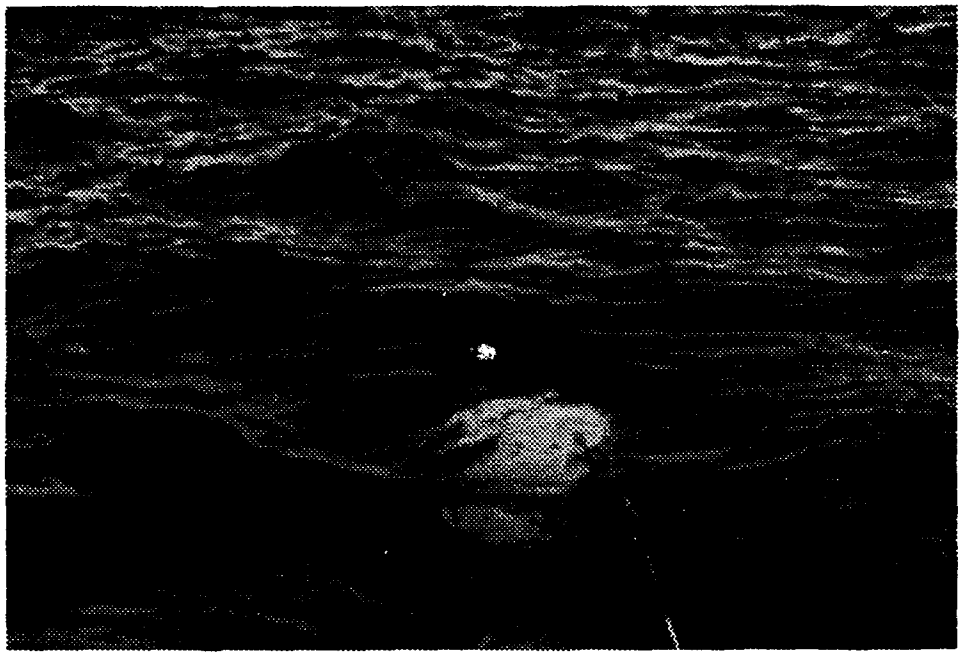


Figure 14. Typical Deep Dive

the platform, with the antenna, was typically underwater for 1.5 - 3 minutes. This would be similar to a large wave crashing over the AUV. Figure 15 is a bar graph of three minutes of

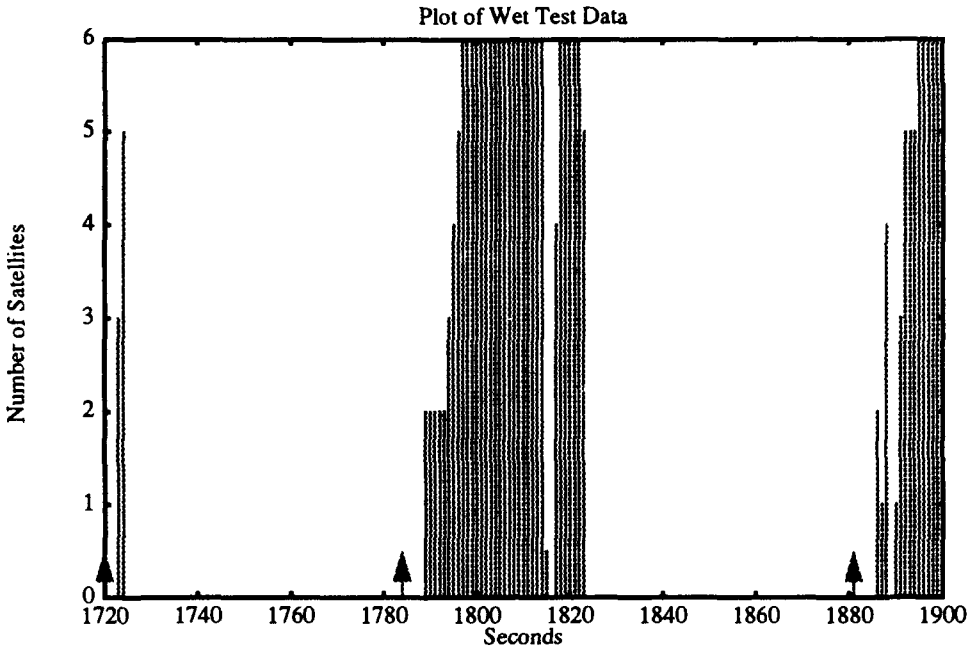


Figure 15. Typical Deep Dive Tracking Results

Represents Antenna Surfacing

the recorded sea test data. The time is in elapsed seconds beginning when the system was put in the water. Each bar shows the number of satellites that the receiver was tracking during that second of the interval. The bars with an arrow indicate that the antenna has surfaced. This graph shows two dives of about 1 minute each. The time to reacquire three satellites after surfacing was only 10 seconds in one case and 7 seconds in the other. The time to reacquire all six satellites was only 13 and 14 seconds.

Short, shallow dives of 1 - 3 feet were performed and a sample can be seen in Figure 16, where the typical time obscured was 20 - 60 seconds. This would be an example of a typical ocean wave, during the transit phase. An impulse graph of a typical shallow dive is shown in Figure 17, where it typically takes less than 10 seconds to reacquire at least three satellites after first fix. From this graph of the recorded data, the second interval can be used as a typical short, shallow dive. The antenna was submerged for 40 seconds. It took 6 seconds after surfacing to track three satellites.

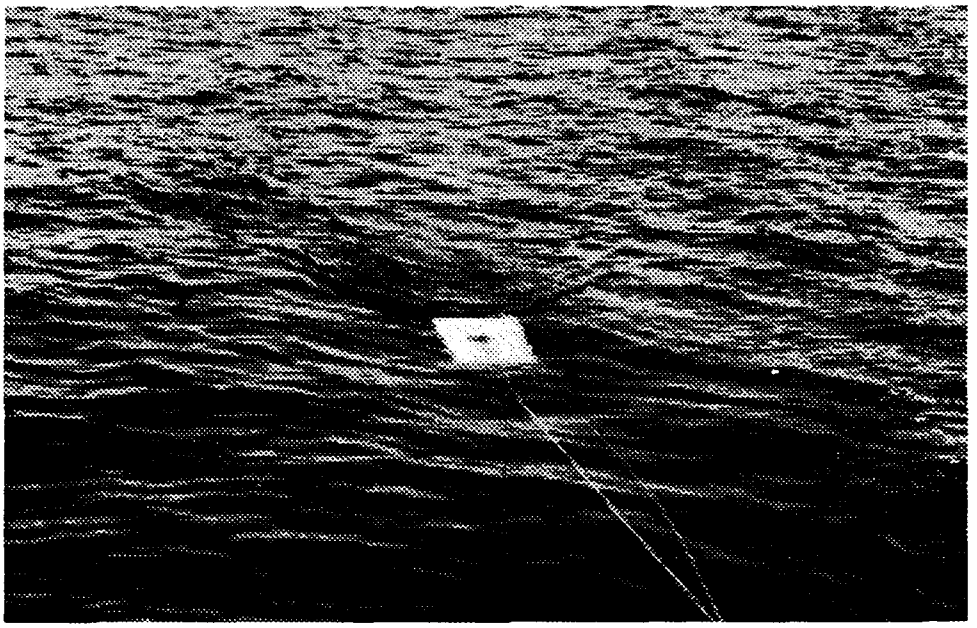


Figure 16. Typical Shallow Dive

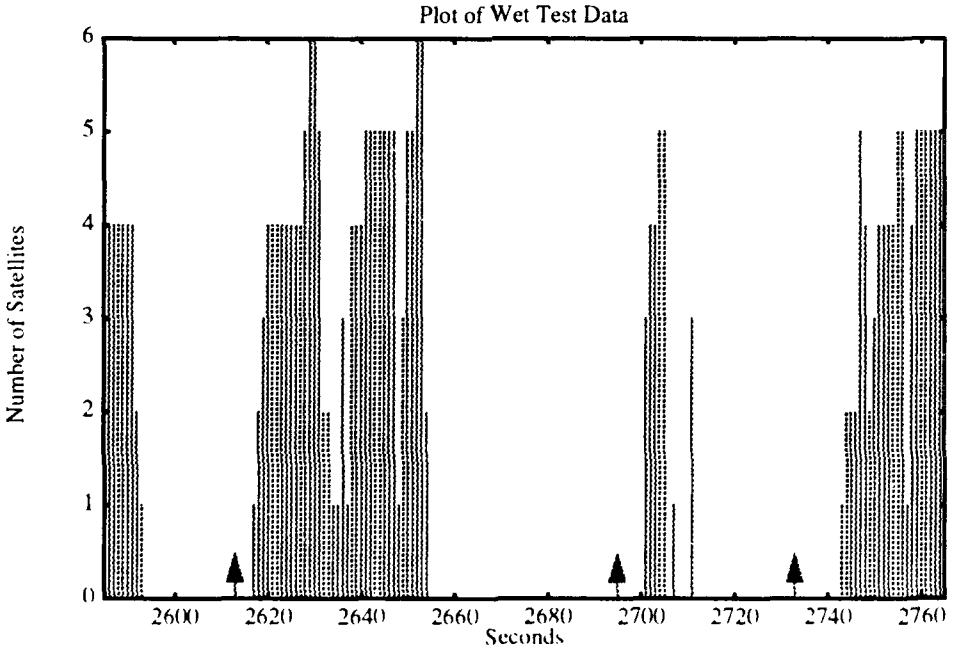


Figure 17. Typical Shallow Dive Tracking Results

▲ Represents Antenna Surfacing

Finally, very short, very shallow repetitive dives were performed that would cover the antenna for just 2 - 5 seconds. An example of the bar graph of several short, shallow dives is shown in Figure 17. This graph shows that with a thin layer of water over the

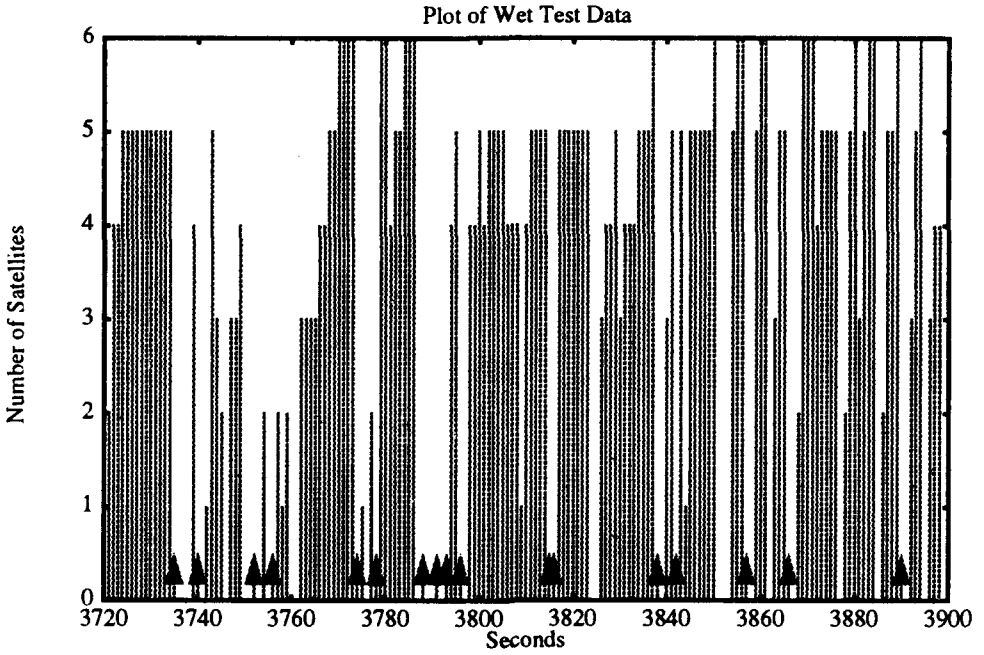


Figure 18. Typical Very Short, Shallow Dive Tracking Results

Represents Antenna Surfacing

antenna, it only takes 2 - 4 seconds to reacquire at least three satellites, if lock was ever lost at all during the submersion. Many times the receiver was able to continue tracking one or two satellites the entire time the water covered the antenna. This confirms that the GPS receiver can reacquire satellites quickly after the antenna has been covered by typical wave action.

Since this test was performed with a flat antenna, a more curved antenna would be expected to provide even better results, if the mission could allow the additional risk of detection. Again, as was stated in relation to the bench test results, the satellites that remained fixed, and that were the first to reacquire during maneuvering, were the satellites with the highest elevation. Having at least one satellite overhead during the mission speeds up the first fix, which provides current ephemeris data for the receiver to find the other

satellites. Having more than one satellite overhead will greatly improve the acquisition time of the SANS.

This sea test was designed with full knowledge of the limitations involved. Determining when the antenna was completely submerged was difficult due to the distance from the boat to the antenna. We were unable to determine the exact depth of the water covering the antenna. Therefore this test was primarily a subjective observation of the ability of the receiver to continue tracking satellites under typical ocean conditions. As a way of quantifying the data from the test, a graph of the sea test is shown in Figure 19. This

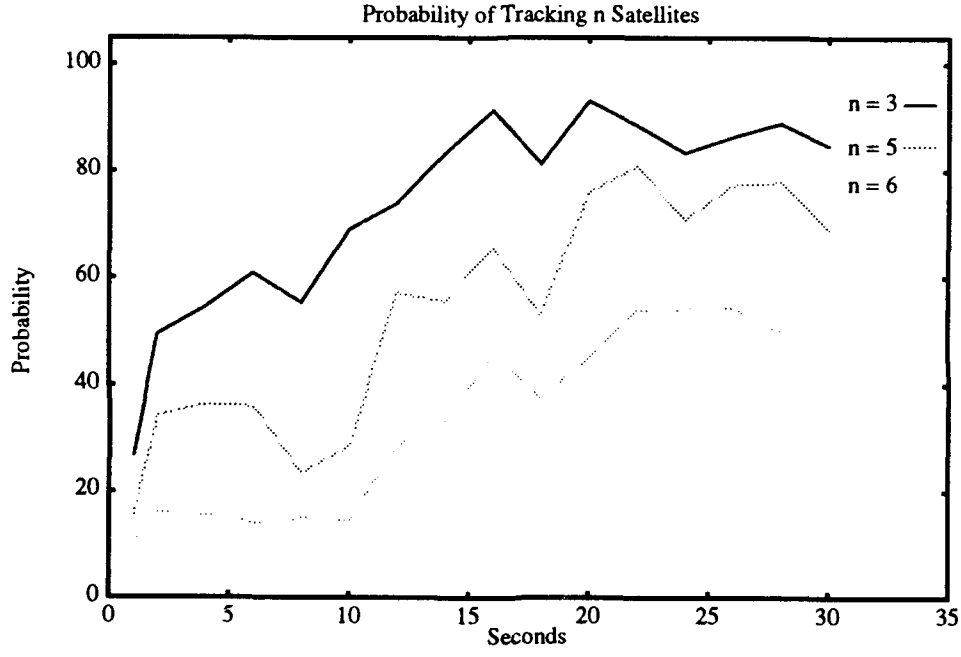


Figure 19. Probability of Tracking n Satellites

graph shows the probability that the receiver was tracking at least n satellites, from n = 3 to n = 6, at any number of seconds after the antenna surfaced. These results show that, throughout the entire test, regardless of the submersion time, within 30 seconds after surfacing, there was over an 80% chance that the receiver was tracking at least three satellites. Therefore, a mission mapping phase that requires the AUV to remain submerged for up to 30 minutes, then allows for a surface time of 30 seconds should be attainable and should not be severely affected by normal wave action. A mission transit phase that

requires even shorter submerged times, and allows 30 second surface times, should be easily obtainable.

E. SIMULATION RESULTS

When the AUV finds an object of interest, it will be required to surface in order to obtain a GPS fix. This fix will give the location of the AUV on the surface, but to determine the location of the submerged object, as shown in Figure 1, it is necessary to calculate the horizontal distance travelled using

$$\Delta H = \frac{\Delta Z}{\tan\theta} \qquad \text{Eq 9}$$

where ΔH is the horiz\ ntal distance travelled, ΔZ is the change in depth, and θ is the climb angle. In the SANS, the depth change is measured by the depth transducer. As stated in Chapter III, using an accelerometer to determine the climb angle of the AUV would allow for removal of the gyroscopes that were evaluated in [STEV93]. The gyroscopes were determined to be unacceptable for this mission due to the constraint on the climb angle of the AUV required to prevent the gyros from tumbling.

In the revised system concept, the accelerometer would be mounted to the navigation system along the longitudinal body axis of the AUV, because additional errors would be introduced if the velocity vector of the AUV deviates from the accelerometer measurement axis. As shown in Equation 2 through Equation 5 in Chapter III, the apparent climb angle of the AUV is calculated using the sensed acceleration and gravity, where

$$\sin(apparent - \theta) = sensed - acceleration/g \qquad \text{Eq 10}$$

and the error between the sin of the actual climb angle and the sin of the apparent climb angle is:

$$a/g = \sin(apparent - \theta) - \sin(\theta) \qquad \text{Eq 11}$$

The horizontal error between the actual location and the expected location is calculated using Equation 12, where ΔZ is the change in depth of the AUV.

$$horizontal - error = \Delta Z\left(\frac{1}{\tan(\theta)} - \frac{1}{\tan(apparent - \theta)}\right) \qquad \text{Eq 12}$$

From these calculations, we can determine what the expected error will be from the accelerometer, given the actual climb angle, the change in depth and the acceleration, a, used by the AUV.

These equations were included in a computer simulation of an AUV with a SANS. By giving the AUV climb angle and an acceleration profile, the simulation program function "calculate-error" will display the path of the AUV diving or surfacing, and calculate the horizontal error in meters. The acceleration profile is a list of lists, made up of the longitudinal acceleration and turning radius of the AUV, and the start and stop time of that segment in seconds. Each change in acceleration or turning radius requires a separate list. The program will display a window with a "wire-frame" model of the AUV, which is redrawn at the end of each list from the acceleration profile, at the current location of the AUV. An X will mark the location where the AUV is estimated to be, based on the calculations.

In a climb, initially the AUV would accelerate to its normal surfacing speed, maintain constant velocity for some time, then decelerate such that the velocity would be back to zero on the surface. Initially, during the accelerating portion of the climb, the horizontal error will be negative, that is the computed location is behind the actual location. During the time of constant velocity, since there is no acceleration, the apparent climb angle is equal to the actual climb angle, and there is no horizontal error. During the decelerating portion of the climb, the horizontal error will be positive, and the estimated location will be forward of the actual location. The error caused during deceleration will nearly cancel out the error caused during acceleration. The total horizontal error is a sum of the horizontal error created during each time interval and the error will be decreased significantly for the entire climb. This method will only be required for computing the horizontal distance travelled while climbing, so the errors will always be reduced by this combination of acceleration and deceleration.

A number of simulations were run in order to provide the horizontal errors expected from the accelerometer under various conditions. These are shown below in Table 5. An acceleration of 0.98 m/sec^2 was used for these calculations, with a maximum velocity of

4.9 m/sec. The horizontal error is reduced further by reducing the acceleration and/or the maximum velocity of the AUV.

70 m Climb		20 m Climb	
Climb Angle	Horizontal Error	Climb Angle	Horizontal Error
20°	-2.27 m	10°	-12.12 m
25°	-1.43 m	12°	-7.35 m
30°	-0.98 m	15°	-4.28 m
45°	-0.35 m	20°	-2.27 m
60°	0.51 m	30°	-0.98 m

Table 5. Sample Horizontal Error Calculations

A sample of the simulation display is shown in Figure 21, which shows the path of the actual location of the AUV, and the Xs show the estimated location based on the calculations shown in this chapter. This display is a side view of the AUV during the climb. In this sample acceleration profile, the AUV will accelerate at 0.98 m/sec^2 for the first 5 seconds of the climb, reaching a maximum velocity of 4.9 m/sec. Then the AUV will maintain a constant velocity for 35.5 seconds, and finally, will decelerate for 5 seconds at -0.98 m/sec^2. This profile is displayed in Figure 20.

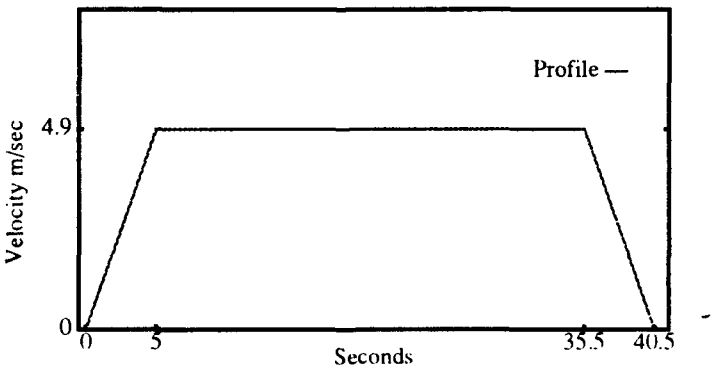


Figure 20. Profile of Sample AUV Surfacing

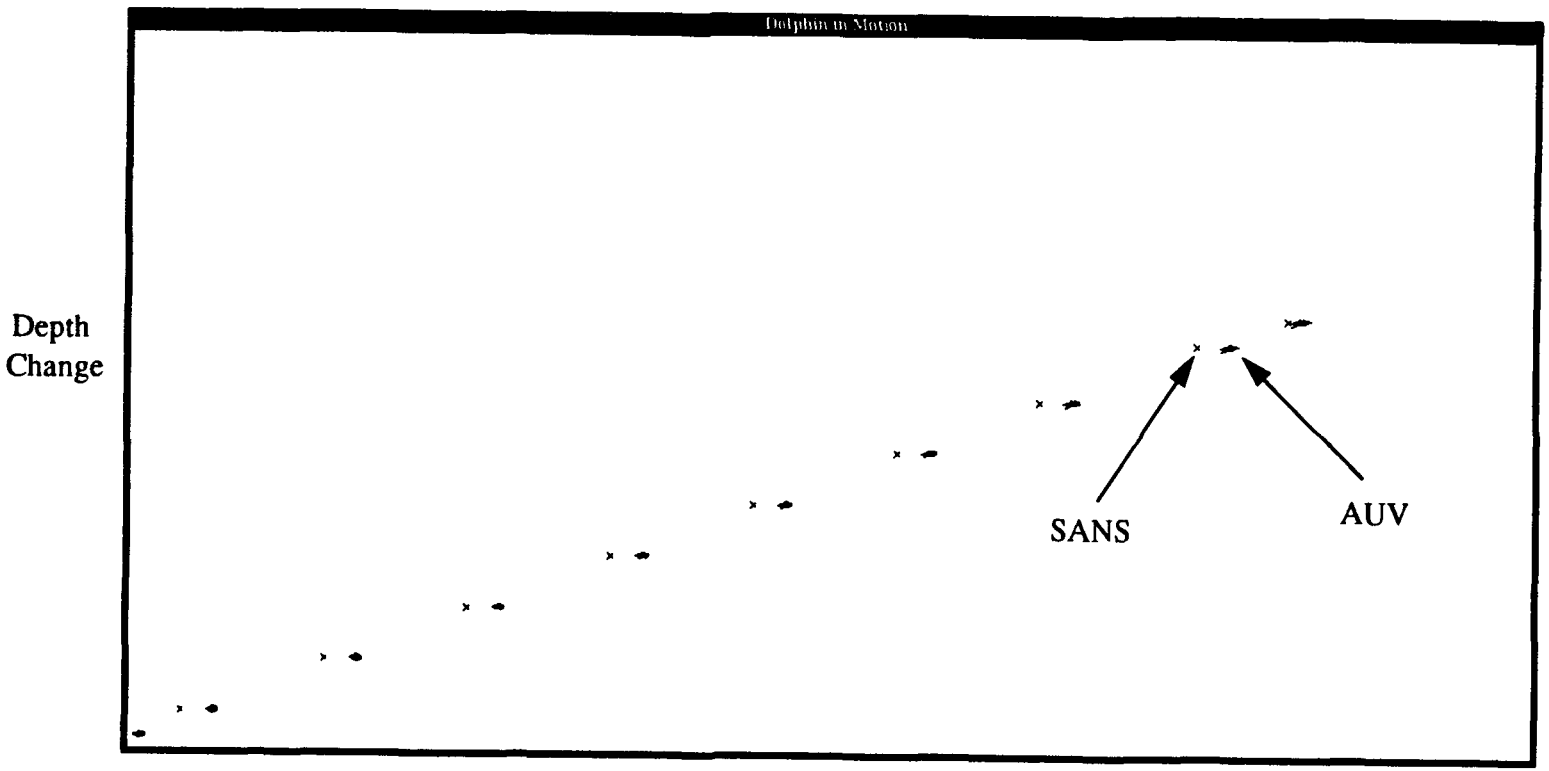


Figure 21. Simulation of 70 m Climb with 20^{o} Climb Angle Showing Accumulate Horizontal Error

These calculations show that the total horizontal error introduced by using an accelerometer to calculate the climb angle, and determine the horizontal distance travelled, is minor, and will not significantly effect the ability of the SANS to obtain an accuracy of 10 meters rms or better.

F. SUMMARY

This chapter provides an explanation of the experimental tests that were developed and performed to determine how a navigation system using GPS would perform under the conditions required by the mission as described in Chapter III. It also explains the computer simulation used to determine the accuracy of the measurement of horizontal distance travelled by the AUV from a submerged object to the surface. The distance is calculated based on using an accelerometer to determine the climb angle of the AUV.

Static testing was done to confirm the acquisition time and accuracy of the Motorola GPS receiver required to track the first satellite, and determine the time required to track three satellites. This provides information needed to assess the ability to use differential processing. The static tests show that the receiver is able to acquire a first fix, with three satellites tracked, within the required 30 seconds, when the receiver is off for 30 minutes or less. The accuracy of the receiver is well within the 100 meter accuracy level of Standard Positioning Service (SPS).

The mission of the AUV requires that the SANS and the antenna mounted on the AUV are as undetectable as possible in order for the mission to remain covert. This means that the antenna must not protrude out of the water more than a few inches. Therefore, normal wave activity of the ocean will repeatedly cover the antenna with a thin layer of water for short periods of time whenever the AUV is surfaced. To determine whether this layer of water would degrade the GPS signal beyond use, a bench test was performed. This bench test showed that the receiver was able to continue tracking at least three satellites while the antenna is continuously covered with 3 to 5 mm of sea water. When the water obstructs the signal for a short time, and is removed, such as a wave would, the receiver is able to reacquire additional satellites within 3 - 5 seconds. The antenna used for testing was

a flat antenna which would allow water to sit on it. Using a fin shaped antenna would improve the ability of the receiver to continue tracking since less of the antenna would be covered at any time. Also, the testing showed that the higher the elevation of the visible satellites, the better the receiver was able to continue tracking, even with 5 to 11 mm of water covering the antenna.

As a final test to demonstrate the ability of the SANS to operate in its proposed environment, an actual sea test was performed on the antenna and receiver. This sea test used a wooden tow-body, towed behind a sail boat, that could be controlled to float on the surface of the water, be towed just under the surface, or be made to dive to various depths. The antenna was mounted on the float, and connected to the receiver with RG-213 cable. The entire test was recorded on video tape and the position and range data from the receiver was written to a file for evaluation. During this sea test, various lengths and depths of dives were performed as samples of the various types of waves that could effect the GPS signal during the transit phase of the AUV mission. Each of these dives, from very short to long, deep dives, showed that overall, the receiver was able to reacquire the GPS signal and track at least three satellites during over 80% of the time. The receiver was typically able to reacquire three or more satellites within 10 to 12 seconds after surfacing.

A computer simulation was used to determine the error introduced by using the sensed acceleration from an accelerometer in the SANS to calculate the climb angle of the AUV when climbing from a submerged object of interest back to the surface. This climb angle was then used to calculate the horizontal distance travelled from the object to the surface, to determine the actual location of the object. The simulation results shown in Table 5 are a sample that illustrate that the horizontal error introduced is minor. Because the AUV is accelerating and decelerating during the climb, the positive and negative location errors tend to cancel each other, and the horizontal error would not significantly reduce the accuracy of the SANS. Additionally, by reducing the acceleration and maximum velocity of the AUV, the horizontal error is reduced further.

This combination of experimental testing and simulation show that the hardware configuration as described in Chapter IV is able to meet the requirements for the transit

phase and the mission phase. The AUV is able to remain covert, spend no more than 30 seconds on the surface during each interval, have the antenna exposed only a few inches above the surface, allow a submerged time of up to 30 minutes, and obtain an overall accuracy of 10 meters rms or better. These tests also show that the effects of normal waves of the ocean should not significantly degrade the capability of the navigation system. The simulation results show that replacing gyroscopes with an accelerometer should not degrade the capability of the system either.

VI. LISP SIMULATION CODE DESCRIPTION

A. INTRODUCTION

A very basic computer simulation of an AUV and the SANS from this research was designed in order to improve the ability of researchers to test various theories involved in the design of the SANS. Replacing the gyroscopes in the SANS used to measure the climb angle of the AUV when surfacing, with an accelerometer was proposed by Prof. McGhee in [MCGH93]. This change would require using the acceleration sensed by the accelerometer to calculate the climb angle of the AUV. As discussed previously, this method would introduce error into the final computation of the horizontal distance travelled by the AUV from a submerged object back to the surface. In order to reduce the time and effort required to compute this error, the equations needed were included in the software for the simulation. By using the simulation, with the required parameters, the expected horizontal error can be quickly and easily determined.

The simulation was written in CLOS (Common LISP Object System), which provides an object oriented programming design. Because the SANS is made up of many parts, an object oriented programming language is preferred. CLOS allows the AUV and the SANS to be represented as objects and classes, and each sensor within the SANS has a slot which contains the measurements for that sensor. Another advantage to using CLOS is that it is an interpretive language. This makes it very simple for the programmer and user to determine at any time what the status of the objects and their slot values are. This chapter explains the Hunter simulation code. The basis for this simulation was an Aquarobot walking machine simulation that was written by Prof. McGhee, and is described in [DAVI93].

B. CLASS AND OBJECT HIERARCHY

Object oriented programming allows a complex system to be designed as a group of elementary components that are linked together as required by the system. This is done

using classes and objects. Classes are the blueprints or forms for a component, while objects are the components that can be manipulated using the software program. [DAVI93]

A class provides the template for creating many objects. The class provides the correct information, in each slot, required for that class. A class can inherit from multiple superclasses, giving it the information from the slots of all of it's superclasses. An abstract class is designed as a template which can contain information needed by multiple subclasses. This multiple inheritance reduces the amount of duplicated information. An abstract class cannot have an object instantiated. Only concrete classes have objects instantiated from them. [DAVI93]

An object is made from the instantiation of a concrete class. All objects from the same class have the same slots, although they can all be manipulated individually after instantiation. Each object is created by making an instance of the class, and each object has its own name within the program.

C. HUNTER CLASS DEFINITION CODE

The *rigid body* class is used as a superclass for the Hunter simulation. The *hunter* is a subclass of *rigid body* made up of an AUV and a SANS. The AUV, called *dolphin* in this simulation, is made up of a *dolphin body*, a *dorsal fin*, a *right pectoral fin*, a *left pectoral fin*, and a *tail fin*, all of which are separate subclasses of *rigid body*. The fins are attached to the body using the *link* class. The *SANS* is also attached to the body using the *link* class, but the *SANS* uses *link0* and *link1*, and the fins use *link2* and *link3*. The class hierarchy is shown in Figure 22.

Classes in CLOS are written according to a template with mandatory and optional information. The CLOS classes allow the use of slots for the items within a class. These slots are defined and initialized using the *:initarg* or *:initform* command. Figure 23 shows a sample of code for the dependent objects instantiated within the abstract *dolphin-fin* class. The *make-instance* command is used to instantiate objects, such as *link2*. Functions related to the class are defined outside of the class definition in *defmethods*. The "initialize-fin"

function requires a *fin* and a *dolphin-body* as input. Functions can be used to change slot values, call other functions, and assign variables. [DAVI93]

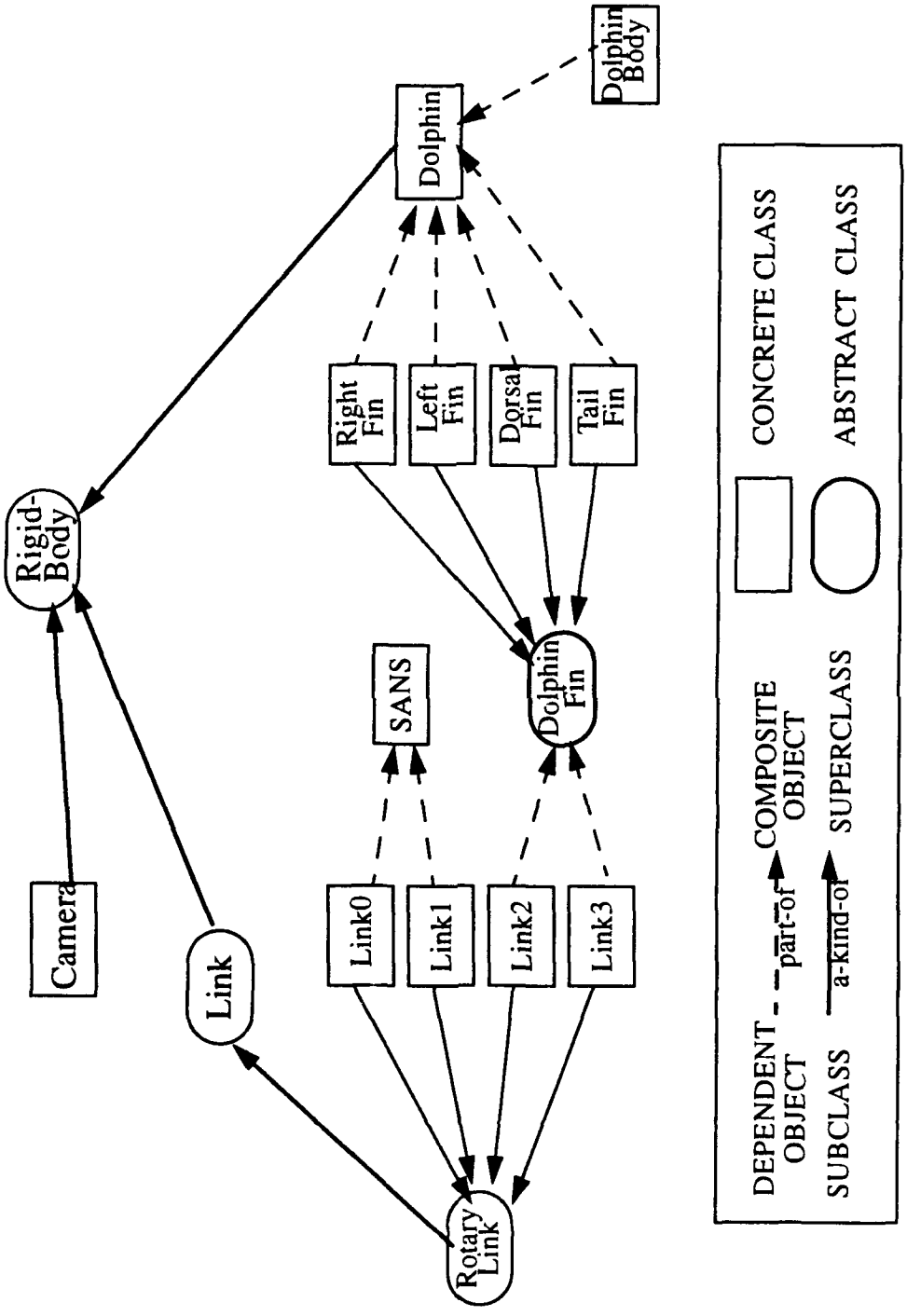


Figure 22. CLOS Hunter Class Hierarchy

```
(defclass dolphin-fin ()
 ((fin-attachment-angle
   :initarg :fin-attachment-angle
   :accessor fin-attachment-angle)
  (link2
   :initform (make-instance 'link2)
   :accessor link2)
  (link3
   :initform (make-instance 'link3)
   :accessor link3)
  (motion-complete-flag
   :initform nil
   :accessor motion-complete-flag)
  (previous-fin-position
   :initform nil
   :accessor previous-fin-position)
  (current-fin-position
   :initform nil
   :accessor current-fin-position)))

(defclass right-fin (dolphin-fin)
 ((link2 :initform (make-instance 'link2
    :link-length 22
    :twist-angle (deg-to-rad 90)
    :inboard-joint-displacement 1))
  (link3 :initform (make-instance 'link3
    :node-list '((0 0 0 1) (0 10 10 1) (44 30 0 1) (0 0 40 1))))))

(defmethod initialize-fin ((fin dolphin-fin) (aqua dolphin))
 (setf (inboard-link (link2 fin)) (body aqua))
 (setf (inboard-link (link3 fin)) (link2 fin))
 (rotate-link (link2 fin) (fin-attachment-angle fin))
 (rotate-link (link3 fin) (inboard-joint-angle (link3 fin)))
 (setf (current-fin-position fin)
     (firstn 3 (first (transform-node-list (link3 fin))))))
```

Figure 23. CLOS Code Excerpt Defining and Implementing Dolphin Fin Kinematics

D. HUNTER OBJECT INSTANTIATION CODE

The object hierarchy of the simulation is shown in Figure 24. It has one top level object, the *hunter*, with the dependent objects as shown.

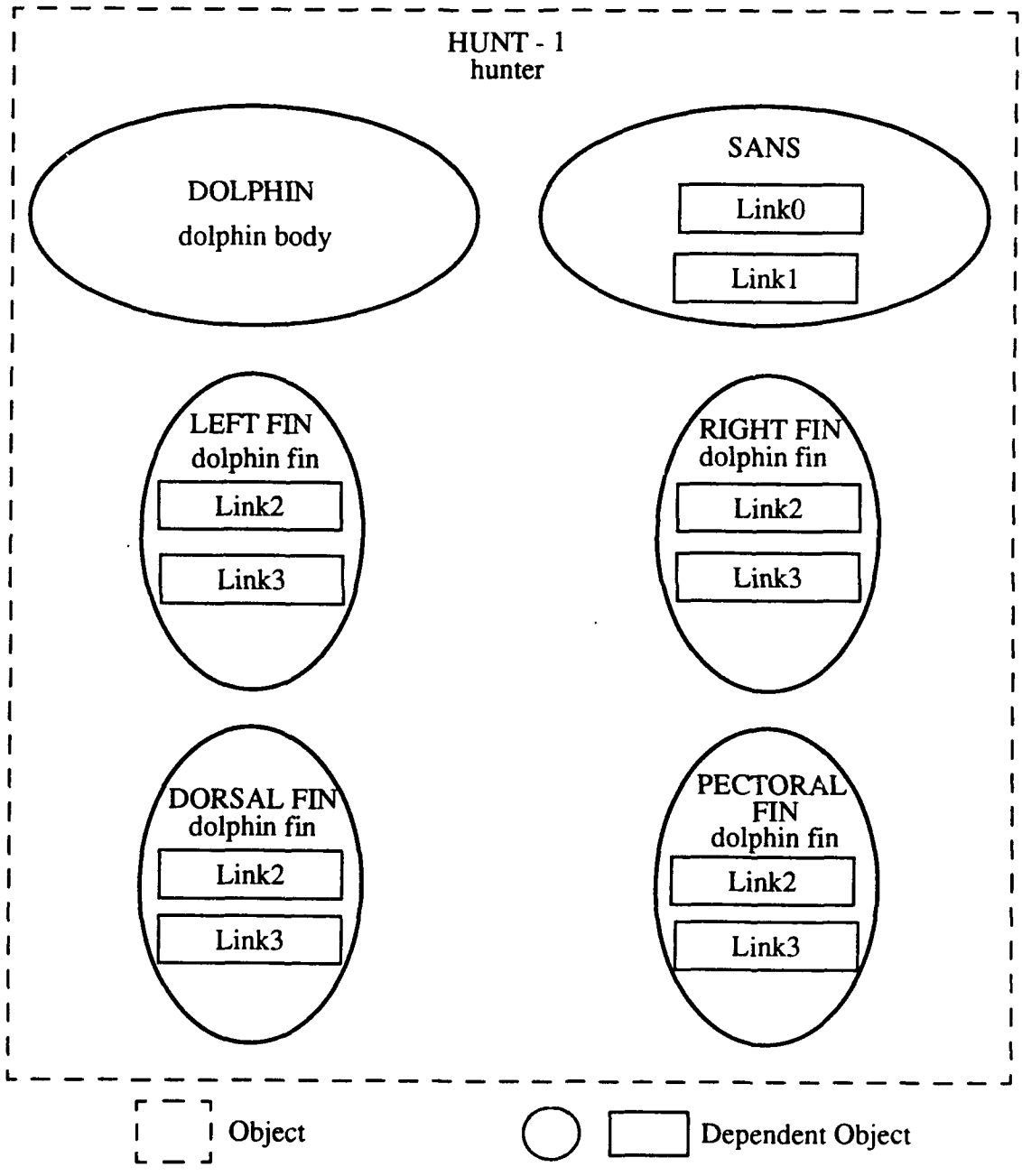


Figure 24. CLOS Object Hierarchy

The hunter simulation creates a *hunter* object by performing the function "make-hunter-picture", as seen in Figure 25. This will make an instance of a *hunter*, with a *dolphin* and a *SANS*, initializes the *hunter* which will initialize its dependent objects, make an instance of an *X* to mark the error location, make an instance of a *camera*, and take a picture of the *hunter*. Even though every *hunter* is created exactly alike, each one has a different name within the program and can be manipulated separately.

```
(defclass hunter ()
  ((dolphin
     :initform (make-instance 'dolphin)
     :accessor dolphin)
   (sans
     :initform (make-instance 'sans)
     :accessor sans)
   (node-list
     :initform '((0 0 0 1))
     :accessor node-list)))

(defmethod initialize ((hunt1 hunter))
  (setf delta-t (get-delta-t (sans hunt1)))
  (initialize (dolphin hunt1))
  (initialize-sans (sans hunt1) hunt1)
  (setf (node-list hunt1) (append (node-list (dolphin hunt1))
  (node-list (link1 (sans hunt1)))))
  (update-velocity-growth-rate hunt1)
  (transform-node-list hunt1))

(defmethod make-hunter-picture ()
  (setf hunt1 (make-instance 'hunter))
  (initialize hunt1)
  (x-picture)
  (setf cam1 (make-instance 'camera))
  (take-picture cam1 hunt1))
```

Figure 25. CLOS Code for Hunter Class

The dependent objects *dolphin* and *SANS*, that are part of the top-level object *hunter*, and their dependent objects, are created automatically within the code when an instance of *hunter* is created. Because they are dependent objects, they cannot be referred

to without referring to them through the *hunter*, and they are all destroyed when the *hunter* is destroyed. A sample of code showing the class definition for the dependent object *dolphin* is shown in Figure 26.

```
(defclass dolphin ()
  ((body
    :initform (make-instance 'dolphin-body)
    :accessor body)
   (right-fin
    :initform (make-instance 'right-fin :fin-attachment-angle (deg-to-rad 90))
    :accessor right-fin)
   (left-fin
    :initform (make-instance 'left-fin :fin-attachment-angle (deg-to-rad -90))
    :accessor left-fin)
   (dorsal-fin
    :initform (make-instance 'dorsal-fin :fin-attachment-angle
                (deg-to-rad 90))
    :accessor dorsal-fin)
   (tail-fin
    :initform (make-instance 'tail-fin :fin-attachment-angle (deg-to-rad 0))
    :accessor tail-fin)
   (node-list
    :initform '((0 0 0 1))
    :accessor node-list)))
```

Figure 26. CLOS Code for Dolphin Class

E. SIMULATION CODE

In order to calculate the climb angle of the AUV in the Hunter simulation, a number of functions had to be written. These functions perform the same calculations as those shown in Equation 9 through Equation 12, in Chapter V. A sample of the functions is shown in Figure 27. "Set-accelerometer" performs the calculations in Equation 13 and Equation 14, while "update-horizontal-error" performs the calculation in Equation 15.

$$sensed-acceleration = a + g\sin(\theta) \qquad \text{Eq 13}$$

$$\sin(apparent-\theta) = \sin(\theta) + a/g \qquad \text{Eq 14}$$

$$\Delta horizontal-error = \Delta Z\left(\frac{1}{\tan(\theta)} - \frac{1}{\tan(apparent-\theta)}\right) \qquad \text{Eq 15}$$

```
(defmethod set-accelerometer ((hunt1 hunter))
  (setf (sensed-acceleration (sans hunt1))
    (+ (first (velocity-growth-rate (sans hunt1)))
    (* *gravity* (sin (deg-to-rad (climb-angle (sans hunt1)))))))
  (setf apparent-climb-angle
    (rad-to-deg (asin (+ (sin (deg-to-rad (climb-angle (sans hunt1))))
                         (/ (first (velocity-growth-rate (sans hunt1))) *gravity*)))))
  (update-horizontal-error hunt1))

(defmethod update-horizontal-error ((hunt1 hunter))
  (setf (horizontal-error (sans hunt1))
    (* (delta-depth (sans hunt1))
    (- (/ 1 (tan (deg-to-rad (climb-angle (sans hunt1)))))
    (/ 1 (tan (deg-to-rad apparent-climb-angle)))))))
```

Figure 27. CLOS Code Excerpt Showing Functions Used to Calculate Climb Angle

In order to compute the estimated horizontal error introduced by using an accelerometer to determine the climb angle of the AUV, the hunter simulation requires the user to assign an acceleration profile and provide the climb angle used for surfacing. An example of this is shown in Figure 28. The acceleration profile is made up of a list of lists.

```
(setf profile-2070 '((0.98 -4 0 5) (0  -4 5 10) (0 -4 10 15) (0 -4 15 20)
                     (0 -4 20 25) (0 -4 25 30) (0 -4 30 35) (0 -4 35 40.5)
                     (-0.98 -4 40.5 45.5)))

(calculate-error-demo '-20 profile-2070)
```

Figure 28. Sample User Input Required to Calculate Horizontal Error

The first element in the list, 0.98 in this example, is the acceleration, in m/sec^2, of the AUV during that time period. The second element is the turn radius of the climb, -4 m. The third and fourth elements are the start and stop times in seconds for that portion of the profile. In this sample the AUV will accelerate at 0.98 m/sec^2 for the first 5 seconds of the climb,

reaching a maximum velocity of 4.9 m/sec. Then the AUV will maintain a constant velocity for 35.5 seconds, after which it will decelerate for 5 seconds at -0.98 m/sec^2. The function calculate-error-demo provides the climb angle for the climb of -20^0, and the desired acceleration profile. In this example the AUV climbed 68 m. This simulation uses the standard coordinate system used in aerodynamics, with positive z being down, and negative z being up.

F. GRAPHICS DISPLAY

The graphical display in CLOS is created using an object called a *camera.* It was created as a debugging tool [DAVI93], and in this simulation it can be used to visually demonstrate the horizontal error at each stage of the climb. The wire-frame diagram of the *hunter*, with a *dolphin* and a *SANS* is shown in Figure 29 as it would appear on the screen.

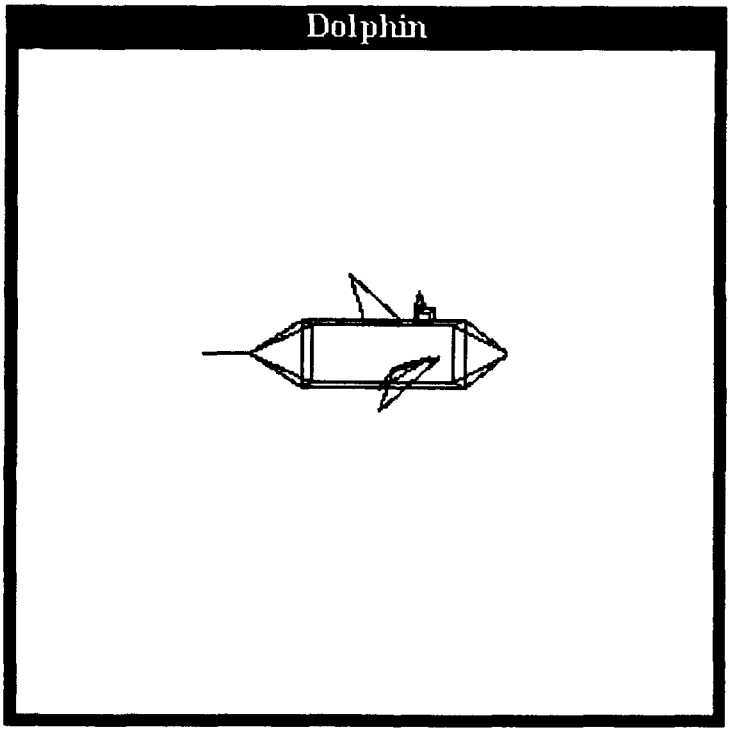


Figure 29. Graphical Display of a Hunter

The design of the wire-frame diagrams of the objects is determined by the node list and

polygon list for each object. These are used to draw the object. Figure 30 is an example of the node list used to draw the *SANS*. Each point is given in centimeters, and the simulation bases the local coordinate system origin on the center of the *dolphin body*.

```
(defclass sans (rigid-body)
 ((link1
   :initform (make-instance 'link1              ;in centimeters
   :node-list '((0 0 0 1)
      (25 10 -22 1) (37 10 -22 1) (37 10 -30 1) (25 10 -30 1)
      (25 -10 -22 1) (37 -10 -22 1) (37 -10 -30 1) (25 -10 -30 1)
      (37 10 -22 1) (37 -10 -22 1) (37 -10 -30 1) (37 10 -30 1)
      (25 10 -22 1) (25 -10 -22 1) (25 -10 -30 1) (25 10 -30 1)

      (25 3 -30 1) (25 -3 -30 1) (25 -3 -33 1) (25 3 -33 1)
      (31 3 -30 1) (31 -3 -30 1) (31 -3 -33 1) (31 3 -33 1)

      (25 3 -30 1) (31 3 -30 1) (31 3 -33 1) (25 3 -33 1)
      (25 -3 -30 1) (31 -3 -30 1) (31 -3 -33 1) (25 -3 -33 1)
      (25 2 -33 1) (31 2 -33 1) (28 0 -44 1)
      (31 2 -33 1) (31 -2 -33 1) (28 0 -44 1)
      (31 -2 -33 1) (25 -2 -33 1) (28 0 -44 1)
      (25 -2 -33 1) (25 2 -33 1) (28 0 -44 1))

   :polygon-list '((1 2 3 4) (5 6 7 8) (9 10 11 12) (13 14 15 16) (17 18 19 20)
      (21 22 23 24) (25 26 27 28) (29 30 31 32) (33 34 35) (36 37 38)
      (39 40 41) (42 43 44))

   :min-joint-angle '0
   :max-joint-angle '0)
   :accessor link1)))
```

Figure 30. CLOS Code Excerpt Showing Node List of SANS

The camera can be moved to obtain views of the hunter from various angles, and the movement of the dolphin and its fins can be seen on the screen. Because CLOS is an interpreted language, the user can see each change or movement of the objects as they are entered into the command line. A complete listing of all simulation code used in this thesis is provided in Appendix B.

G. SUMMARY

In order to increase the ability to test and evaluate the SANS, a computer simulation of the system was developed, called a *Hunter*. As described in Chapter V, this simulation currently has the capability to determine the estimated errors introduced by using an accelerometer to measure acceleration and thus calculate the climb angle, and the horizontal distance travelled by an AUV from a submerged object to the surface. This would be used to determine the actual location of the object, from the GPS location obtained after surfacing.

The hunter computer simulation is written in CLOS, and is made up of classes and objects as would be logical for the design of a system comprised of an AUV, called *dolphin* in this simulation, and a *SANS*. The *hunter* is the top-level object. The *dolphin* and the *SANS* are comprised of objects as well, and each have slot values that are assigned when created. The slot values in the *SANS* are used to store information that would be available from various sensors in the actual SANS. For example, the *SANS* has a slot called *sensed-acceleration*, which would be a reading from the accelerometer. The class and object hierarchy and sample code for defining and instantiating classes and objects are shown.

The function "calculate-error" uses the series of equations previously shown to calculate the horizontal error expected from the use of an accelerometer in the SANS. Sample code of these equations is provided. The user must provide an acceleration profile for the AUV to use for the simulation of a climb, as well as the climb angle. This will require providing the expected acceleration rate of the AUV, the maximum velocity reached and the expected deceleration rate, as well as the duration of each of these events during the surfacing maneuver.

In order to better explain and understand the expected actions and responses of the AUV and SANS, a graphical display of the objects is included in the simulation. This allows for viewing a simulation of the AUV performing a desired climb.

VII. CONCLUSIONS AND RECOMMENDATIONS

A. CONCLUSIONS

The mission requirements for the Small Autonomous Underwater Vehicle (AUV) Navigation System (SANS) were outlined in Chapter I. The hardware testing performed in this research was done in order to determine whether the mission requirement for having the GPS antenna at the surface of the ocean while limiting visibility would degrade the ability of the GPS system beyond use. The mission does not allow the antenna to protrude out of the water more than a few inches. Therefore, due to normal surface waves, the antenna will usually be wet, and will repeatedly be covered with water for short periods of time throughout the mission of the AUV. This research performed a series of experimental tests that show that the GPS signal is able to penetrate a shallow layer of sea water when it is covering a GPS antenna, and that the normal wave wash from the ocean should not significantly degrade the ability of the system to obtain a GPS fix within 30 seconds of surfacing. During the mission phase, the AUV must be able to submerge for several minutes at a time and still be able to obtain a GPS fix within 30 seconds after surfacing. The hardware testing described in Chapter V show that the GPS system is capable of meeting this requirement.

The miniature gyroscopes used in the SANS design described in [STEV93] needed to be replaced because the AUV climb angle would have to be limited to prevent gyro tumbling. In order to determine whether an accelerometer could be used in the system to determine the climb angle, as described in [MCGH93], extensive error calculations had to be performed. To simplify the process of calculating these errors, a graphical computer simulation was written in CLOS (Common LISP Object System). After finding an object of interest the AUV will be required to surface to obtain a GPS fix. The climb angle of the surfacing is used to determine the position of the object based on the GPS fix. This computer simulation determines the horizontal error that is the difference between the actual location of a submerged object to the position that the SANS sensors expect the object to be. The simulation is run using various surfacing profiles, based on an

acceleration profile that gives the acceleration and turn radius for each second of the surface. The results of this simulation show that the error introduced from the accelerometer do not greatly reduce the accuracy of the SANS, and that it could be used to replace the gyroscopes thereby removing restrictions on the climb angle of the AUV.

All of the hardware components described in this thesis meet or exceed the requirements for the mission. This research concludes that all of the mission objectives can be met using the current SANS design.

B. RECOMMENDATIONS

1. Future Research

Recommendations for future research on the SANS project fall into two categories, software and hardware. The next phase in the software development is to implement and evaluate the software design from [STEV93] on the E.S.P. 8680. The next phase in the hardware development is to test the system using differential GPS, either real-time or with post-processing, to determine the accuracy of the differential system.

Additionally, the new, high performance, low-power TCM1 Electronic Compass Sensor Module developed by Precision Navigation could be included in the SANS. It is based on a triaxial magnetometer system and a biaxial electrolytic inclinometer. Developer specifications show that this system should be able to meet all mission requirements. [PREC94]

2. Future Use of Code

The computer simulation written for this research could be further developed for other purposes. New sensors could be added to the program to determine their expected accuracy. Forces of motion on the rigid body could be added to improve the realism of the simulation. The AUV shape and size can be changed easily, as can the acceleration, velocity and climb angle. These changes could be made to adapt the program to other types of AUV missions.

APPENDIX A: TECHNICAL SPECIFICATIONS

A. DOVATRON ESP-8680 CORE MODULE

Processor:	14 MHz 5-volt 8680 (8086 equivalent)
Serial Port:	RS-232
Keyboard:	Standard controller and direct key switch matrix scanning
Graphics:	CGA (Color Graphics Adapter) or LCD (Liquid Crystal Display)
Memory:	256K x 8 EPROM or Flash Memory 512K or 1MB DRAM (8-bit or 16-bit wide memory path) PCMCIA card slot for up to 16MB Memory
Memory Option:	Expansion board adds up to 16MB DRAM
Bus Interface:	ISA (Industry Standard Architecture)
Form Factor:	1.7" x 5.1" (13.2cm x 4.3cm)
Power Consumption:	Draws from 1 mA (sleep mode) to 350 mA (peak load powering back-lit LCD and peripherals)

B. DOVATRON E.S.P. A TO D CONVERTER

Input channels:	16 single-ended or 8 differential
A/D resolution:	12-bits (1/4096)
Input ranges:	1.25 mV to 10 V
Programmable Gain:	1 - 8000 (one binary and one decade PGA)
Sample Rate:	333 KHz Maximum (single channel)
Resolution:	20 microvolt minimum
Measurement:	AC/DC Coupling, True RMS
RMS Mode:	Accuracy 1% above 20 KHz, 250 mSec response, 100 KHz BW
Other features:	82C54 Three channel programmable timer. One channel available for general use. Offset voltage trimmer.
Power Requirements:	TBD

C. MOTOROLA PVT-6

Receiver Architecture:	6-channel L1 1575.42 MHz
Tracking Capability:	6 simultaneous satellite vehicles
Dynamics:	Velocity : 1000 Knots (514.4 m/sec) Acceleration: 4 g Jerk: 5 m/s^3
Antenna to Receiver Interconnection:	Single coaxial cable (6dB max loss at L1; 1575.42 MHz) Typical length with RG58 coaxial cable; 20 feet (6 meters)
Serial Output:	RS-232C Interface
Accuracy:	Less than 25 meters, SEP (without SA)
Operating Temperature:	$-30^{o}C$ to $+80^{o}C$
Humidity:	95% non-condensing $+30^{o}C$ to $+60^{o}C$
Physical Dimensions:	3.94 x 2.76 x 0.65 in (100 x 70 x 16.5 mm)
Weight:	4.5 ounces (128 grams)
Switched Power:	9 to 16 Vdc or 5 0.25 Vdc
Keep-Alive Power:	4.75 - 16 Vdc; 0.3 mA
Power Consumption (typical):	1.3 W @ 5 Vdc input 1.8 W @ 12 Vdc input
MTBF	:65,000 hours (estimated)
Acquisition Time (Time To First Fix, TTFF)	:Hot: 21 sec. typical TTFF Warm: 53 sec. typical TTFF
Reacquisition Time:	15 sec obscured: < 2.5 sec. typical 30 sec obscured: < 3.5 sec typical 45 sec obscured: < 3.5 sec typical 60 sec obscured: < 3.6 sec typical

D. KVH C100 DIGITAL COMPASS SENSOR

Accuracy:	±0.5° or ±10 mils RMS
Repeatability:	±0.2° or ±5 mils
Resolution:	0.1° or 5 mils
Dip Angle:	±80° Maintains stated accuracy after autocal over ±80° Magnetic Dip Angle
Tilt Angle:	±16° Dev.= ±0.3° RMS ±45° Dev.= ±0.5° RMS
Electrical Power:	Input Voltage: +8 to +20 VDC or +20 to +30 VDC (user selectable)
	Current Drain: 20 mA DC; nominal
Size:	1.80 x 4.50 x 1.10" (4.6 cm x 11.4 cm x 2.8 cm)
Weight:	2.0 ounces (57 grams)
Environmental Performance:	
Operating Temp.:	-22° to +122°F (-30°C to +50°C)
Vibration:	30 minutes random MIL-STD-810
Shock:	Handling shock per MIL-STD-810
Digital Interfaces:	Standard RS232 Bidirectional Serial Data
Analog Outputs:	
Sine/Cosine:	Sine/Cosine output voltage +2.5V ±1.0V
OR	
Linear Voltage:	0 to +3.6VDC into 10K Ohm minimum load

E. OMEGA PX176-100PSIS DEPTH TRANSDUCER

PX176 Specifications at 25°C

PARAMETER	MIN	TYP	MAX	UNITS
Full Scale Output (FSO) @ 25°C	4.90	4.95-5.05	5.10	Vdc
Null Offset @ 25°C	.85	.95-1.05	1.15	Vdc
Linearity (Best Fit)		±.2	±.5	%FSO
Hysteresis		±.25		%FSO
Temperature Error Null 0° to 85°C		±.01	±.02	%FSO/°C
-55° to 0°C +85°C to 105°C		±.02		%FSO/°C
Sensitivity 0° to 85°C		±.01	±.02	%FSO/°C
-55° to 0°C +85°C to 105°C		±.02		%FSO/°C
Stability (1 year)		±1.0		%FSO
Frequency Response		10		kHz
Supply Voltage	9		20	Vdc
Supply Current (Quiecent)		15		mA
Operating Tempurature	-55°C		105°C	

FSO is the voltage change between minimum and rated pressure.
For example: NOM V_0 = 1.00 V @ Null Pressure, NOM V_0 = 6.00 V @ Rated Pressure, FSO = (6.00 - 1.00) = 5.00 V.

F. HUMPHREY LA67-0108-1 LINEAR ACCELEROMETER

Range:	+0.5 to +1.5 G
Undamped Natural Frequency:	11.5 Hz approximately
Damping Factor:	1.3 ± 0.3 of Critical at $+25^{\circ}C$
Potentiometer Resistance:	5000 Ohms ±10%
Power Dissipation:	0.5 W maximum
Noise:	2.0% full scale maximum
Resolution:	0.1% full scale maximum
Accuracy:	±5% of full scale (with light vibration) applicable between 10 and 90% full scale
Static Threshold:	0.15 G maximum
Insulation Resistance:	50 Megohms min. between all isolated circuits and between any terminals and case when measured with 500 Vdc at standard atmospheric conditions
Temperature:	
Non-Operating:	$-80^{\circ}F$ to $+160^{\circ}F$
Operating:	$-65^{\circ}F$ to $+160^{\circ}F$
Acceleration:	15 G in each axis
Shock:	15 G, 11 msec duration
Sinusoidal Vibration:	0.02 inch D.A., 5 to 11 Hz; 0.13 G, 11 to 44 Hz; 2 G, 44 to 500 Hz
Sealing:	Hermetic
Service Life:	25×10^6 cycles minimum
Weight:	4 ounces

APPENDIX B: SIMULATION CODE

```
:** camera.cl **

(require :xcw)

(cw:initialize-common-windows)

(defclass camera (rigid-body)
        ((focal-length
          :accessor focal-length
          :initform 6)
         (posture
          :accessor posture          ; azim elev roll x y z
          :initform (list 0 0 0 -300 0 0))
         (camera-window
          :accessor camera-window
          :initform (cw:make-window-stream :borders 5
                                           :left 100
                                           :bottom 300
                                           :width 900
                                           :height 400
                                           :title "Dolphin"
                                           :activate-p t))
         (H-matrix
          :initform (homogeneous-transform (deg-to-rad -90) 0 0 0 300 0))
         (inverse-H-matrix
          :accessor inverse-H-matrix
          :initform (inverse-H (homogeneous-transform
(deg-to-rad -90) 0 0 0 300 0)))
         (enlargement-factor
          :accessor enlargement-factor
          :initform 30)))

(defun create-camera-1 ()
  (setf camera-1 (make-instance 'camera)))
```

```
; *** Draw picture functions ********************************************

(defmethod take-picture ((camera camera) (body rigid-body))
 (let ((camera-space-node-list (mapcar #'(lambda (node-location)
                                           (post-multiply (inverse-H-matrix camera) node-location))
                                       (transformed-node-list body))))
  (dolist (polygon (polygon-list body))
   (clip-and-draw-polygon camera polygon camera-space-node-list))))

(defmethod erase-camera-window ((camera camera))
 (cw:clear (camera-window camera)))

(defmethod cam1 ()
 (setf cam1 (make-instance 'camera)))

(defmethod clear ((camera camera))
 (cw:clear (camera-window camera)))

(defmethod flush ((camera camera))
 (cw:flush (camera-window camera)))

(defmethod new-picture ((camera camera) (body rigid-body))
 (erase-camera-window camera)
 (take-picture camera body))

(defmethod clip-and-draw-polygon
  ((camera camera) polygon node-coord-list)
 (do* ((initial-point (nth (first polygon) node-coord-list))
     (from-point initial-point to-point)
     (remaining-nodes (rest polygon) (rest remaining-nodes))
     (to-point (nth (first remaining-nodes) node-coord-list)
           (if (not (null (first remaining-nodes)))
             (nth (first remaining-nodes) node-coord-list))))
   ((null to-point)
    (draw-clipped-projection camera from-point initial-point))
  (draw-clipped-projection camera from-point to-point)))
```

```
(defmethod draw-clipped-projection ((camera camera)
                                    from-point to-point)
 (cond ((and (<= (first from-point) (focal-length camera))
             (<= (first to-point) (focal-length camera))) nil)
       ((<= (first from-point) (focal-length camera))
        (draw-line-in-window camera
                             (perspective-transform camera
                                                    (from-clip camera from-point to-point))
                             (perspective-transform camera to-point)))
       ((<= (first to-point) (focal-length camera))
        (draw-line-in-window camera
                             (perspective-transform camera from-point)
                             (perspective-transform camera
                                                    (to-clip camera from-point to-point))))
       (t (draw-line-in-window camera
                               (perspective-transform camera from-point)
                               (perspective-transform camera to-point)))))

(defmethod from-clip ((camera camera) from-point to-point)
 (let ((scale-factor (/ (- (focal-length camera) (first from-point))
                        (- (first to-point) (first from-point)))))
  (list (+ (first from-point)
           (* scale-factor (- (first to-point) (first from-point))))
        (+ (second from-point)
           (* scale-factor (- (second to-point) (second from-point))))
        (+ (third from-point)
           (* scale-factor (- (third to-point) (third from-point)))) 1)))

(defmethod to-clip ((camera camera) from-point to-point)
 (from-clip camera to-point from-point))

(defmethod draw-line-in-window ((camera camera) start end)
 (cw:draw-line (camera-window camera)
               (cw:make-position :x (first start) :y (second start))
               (cw:make-position :x (first end) :y (second end))
               :brush-width 1))
```

```
(defmethod perspective-transform ((camera camera) point-in-camera-space)
 (let* ((enlargement-factor (enlargement-factor camera))
      (focal-length (focal-length camera))
      (x (first point-in-camera-space)) ;x axis is along opical axis
      (y (second point-in-camera-space)) ;y is out right side of camera
      (z (third point-in-camera-space))) ;z is out bottom of camera
  (list (+ (round (* enlargement-factor (/ (* focal-length y) x)))
        150)                ;to right in camera window
      (+ 150 (round (* enlargement-factor (/ (* focal-length (- z)) x)))))))   ;up in camera
window

; *** Position camera functions ****************************************

(defmethod move-camera ((camera camera) azimuth elevation roll x y z)
 (setf (H-matrix camera) (homogeneous-transform azimuth elevation roll x y z))
 (setf (inverse-H-matrix camera) (inverse-H (H-matrix camera))))

(defmethod zoom-camera ((camera camera) zoom-amount)
 (setf (slot-value camera 'enlargement-factor)
  (+ (slot-value camera 'enlargement-factor) zoom-amount)))


; Rotation in x-y plane about origin
(defmethod rotate-camera ((camera camera) angle-increment) ; in degrees
 (let* ((new-position (posture camera))
      (radius (sqrt (+ (* (fourth new-position) (fourth new-position))
                (* (fifth new-position) (fifth new-position)))))
      (heading (atan (fourth new-position)
                (fifth new-position)))
      (angle (deg-to-rad angle-increment))
      (new-heading (+ heading angle)))
  (setf (first new-position) (- (first new-position) angle)
      (fourth new-position) (* radius (sin new-heading))
      (fifth new-position) (* radius (cos new-heading))
      (posture camera) new-position
      (H-matrix camera)
      (homogeneous-transform (first new-position)
                  (second new-position) (third new-position) (fourth new-position)
                  (fifth new-position) (sixth new-position))
      (inverse-H-matrix camera) (inverse-H (H-matrix camera)))))
```

```
; Vertical tilting about origin in a plane perpendicular to x-y plane
; Max tilt (90 or -90 deg) when top or bottom view of x-y plane is achieved
(defmethod tilt-camera ((camera camera) angle-increment) ; in degrees
 (let* ((new-position (posture camera))
     (radius (sqrt (+ (* (fourth new-position) (fourth new-position))
                 (* (fifth new-position) (fifth new-position))
                 (* (sixth new-position) (sixth new-position)))))
     (tilt (atan (sixth new-position)
             (sqrt (+  (* (fourth new-position) (fourth new-position))
                  (* (fifth new-position) (fifth new-position))))))
                                heading (atan (fourth new-position)
               (fifth new-position)))
     (angle (deg-to-rad angle-increment))
     (new-tilt (cond ((< (abs (+ tilt angle)) tilt-limit) (+ tilt angle))
               (t (cond ((minusp (+ tilt angle)) (* -1 tilt-limit))
                    (t tilt-limit))))))
  (setf (second new-position) new-tilt
     (fourth new-position)
     (cond ((= (abs tilt) (abs new-tilt) tilt-limit)
          (fourth new-position))
        (t (* radius (sin heading) (cos new-tilt))))
     (fifth new-position)
     (cond ((= (abs tilt) (abs new-tilt) tilt-limit)
          (fifth new-position))
        (t (* radius (cos heading) (cos new-tilt))))
     (sixth new-position)
     (cond ((= (abs tilt) (abs new-tilt) tilt-limit)
          (sixth new-position))
        (t (* radius (sin new-tilt))))
     (posture camera) new-position
     (H-matrix camera)
     (homogeneous-transform (first new-position)
                  (second new-position) (third new-position) (fourth new-position)
                  (fifth new-position) (sixth new-position))
     (inverse-H-matrix camera) (inverse-H (H-matrix camera)))))

(defun deg-to-rad (angle) (* .017453292519943295 angle))
(defconstant tilt-limit (deg-to-rad 89.9))
```

```
(defun three-cameras ()
 (setf cam1 (make-instance 'camera))
 (setf (camera-window cam1) (cw:make-window-stream
                    :borders 5
                    :left 900
                    :bottom 0
                    :width 300
                    :height 300
                    :title "Dolphin Cam1"
                    :activate-p t))

 (setf cam2 (make-instance 'camera))
 (setf (camera-window cam2) (cw:make-window-stream
                    :borders 5
                    :left 600
                    :bottom 0
                    :width 300
                    :height 300
                    :title "Dolphin Cam2"
                    :activate-p t))
 (setf cam3 (make-instance 'camera))
 (setf (camera-window cam3) (cw:make-window-stream
                    :borders 5
                    :left 300
                    :bottom 0
                    :width 300
                    :height 300
                    :title "Dolphin Cam3"
                    :activate-p t))
 (move-camera cam1 0 (deg-to-rad -90) 0 0 0 -400) ; top view
 (move-camera cam2 0 0 0 -400 0 0)    ; rear view
 (move-camera cam3 (deg-to-rad -90) 0 0 0 400 0)) ; side view

(defmethod take-three ()
 (take-picture cam1 hunt1)
 (take-picture cam2 hunt1)
 (take-picture cam3 hunt1))

(defmethod three-new-pictures ()
 (clear-three)
 (take-three))
```

```
(defmethod take-three-dolphins ()
  (take-picture cam1 flip)
  (take-picture cam2 flip)
  (take-picture cam3 flip))

(defun clear-three ()
  (cw:clear (camera-window cam1))
  (cw:clear (camera-window cam2))
  (cw:clear (camera-window cam3)))

(defun flush-three ()
  (cw:flush (camera-window cam1))
  (cw:flush (camera-window cam2))
  (cw:flush (camera-window cam3)))

(defmethod zoom1 ((camera camera))
   (move-camera camera (deg-to-rad -90) 0 0 0 20 0))

(defmethod zoom2 ((camera camera))
   (move-camera camera (deg-to-rad -90) 0 0 0 15 0))

(defmethod zoom3 ((camera camera))
   (move-camera camera (deg-to-rad -90) 0 0 0 10 0))

(defmethod zoom1-three ()
  (move-camera cam1 (deg-to-rad -90) 0 0 0 20 0)
  (move-camera cam2 0 0 0 -20 0 0)
  (move-camera cam3 0 (deg-to-rad -90) 0 0 0 -20))

(defmethod zoom2-three ()
  (move-camera cam1 (deg-to-rad -90) 0 0 0 15 0)
  (move-camera cam2 0 0 0 -15 0 0)
  (move-camera cam3 0 (deg-to-rad -90) 0 0 0 -15))

(defmethod zoom3-three ()
  (move-camera cam1 (deg-to-rad -90) 0 0 0 10 0)
  (move-camera cam2 0 0 0 -10 0 0)
  (move-camera cam3 0 (deg-to-rad -90) 0 0 0 -10))
```

```
(defmethod zoom-three ()
  (move-camera cam1 0 (deg-to-rad -90) 0 0 0 -200)
  (move-camera cam2 0 0 0 -200 0 0)
  (move-camera cam3 (deg-to-rad -90) 0 0 0 200 0))

; *** Auxiliary functions **************************************************

(defun kill ()
  (cw:kill-common-windows))

(defun reset-windows ()
  (kill)
  (cw:initialize-common-windows))
```

```
;** dolphin.cl **

(defclass dolphin ()
        ((body
          :initform (make-instance 'dolphin-body)
          :accessor body)
         (right-fin
          :initform (make-instance 'right-fin
                    :fin-attachment-angle (deg-to-rad 90))
          :accessor right-fin)
         (left-fin
          :initform (make-instance 'left-fin
                    :fin-attachment-angle (deg-to-rad -90))
          :accessor left-fin)
         (dorsal-fin
          :initform (make-instance 'dorsal-fin
                    :fin-attachment-angle (deg-to-rad 90))
          :accessor dorsal-fin)
         (tail-fin
          :initform (make-instance 'tail-fin
                    :fin-attachment-angle (deg-to-rad 0))
          :accessor tail-fin)
         (node-list
          :initform '((0 0 0 1))
          :accessor node-list)))
```

```
(defclass dolphin-body (rigid-body)
        ((location
          :initform '(0 0 0))
         (velocity
          :initform '(0 0 0 0 0 0))
         (velocity-growth-rate
          :initform '(0 0 0 0 0 0))
         (mass
          :initform 400)
         (node-list
          :initform                ; in centimeters
          '((0 0 0 1)
            (55 22 -22 1) (55 22 22 1) (-50 22 22 1) (-50 22 -22 1) ;body
            (55 -22 -22 1) (55 -22 22 1) (-50 -22 22 1) (-50 -22 -22 1) ;body
            (55 -22 -22 1) (55 -22 22 1) (55 22 22 1) (55 22 -22 1) ;body
            (-50 -22 -22 1) (-50 -22 22 1) (-50 22 22 1) (-50 22 -22 1) ;body
            (55 22 22 1) (90 0 0 1) (55 22 -22 1) (90 0 0 1) ;nose
            (55 -22 22 1) (90 0 0 1) (55 -22 -22 1) (90 0 0 1) ;nose
            (-50 22 22 1) (-90 0 0 1) (-50 22 -22 1) (-90 0 0 1) ;end
            (-50 -22 22 1) (-90 0 0 1) (-50 -22 -22 1) (-90 0 0 1))) ;end

         (polygon-list
          :initform '((1 2 3 4) (5 6 7 8) (9 10 11 12) (13 14 15 16)
                      (17 18) (19 20) (21 22) (23 24) (25 26)
                      (27 28) (29 30) (31 32)))
         (H-matrix
          :initform (homogeneous-transform 0 0 0 0 0 0))
         (acceleration-profile
          :initform '(((* (- 3) (cos t)) (* (- 3) (sin t)) 0 2)))))

(defmethod initialize ((aqua dolphin))
  (initialize (body aqua))
  (initialize-fin (right-fin aqua) aqua)
  (initialize-fin (left-fin aqua) aqua)
  (initialize-fin (dorsal-fin aqua) aqua)
  (initialize-fin (tail-fin aqua) aqua)
  (setf (node-list aqua) (append (node-list (body aqua))
                                 (node-list (link3 (right-fin aqua)))
                                 (node-list (link3 (left-fin aqua)))
                                 (node-list (link3 (dorsal-fin aqua)))
                                 (node-list (link3 (tail-fin aqua))))))
```

```
(defun aqua-picture ()
 (setf flip (make-instance 'dolphin))
 (initialize flip)
 (setf cam1 (make-instance 'camera))
 (take-picture cam1 flip))

(defmethod take-picture ((camera camera) (aqua dolphin))
 (take-picture camera (body aqua))
 (take-picture camera (right-fin aqua))
 (take-picture camera (left-fin aqua))
 (take-picture camera (dorsal-fin aqua))
 (take-picture camera (tail-fin aqua)))

(defmethod move-and-flap-incremental ((aqua dolphin) increment-list)
 (move-incremental (body aqua) (first increment-list))
 (move-incremental (right-fin aqua) (second increment-list))
 (move-incremental (left-fin aqua) (third increment-list))
 (move-incremental (dorsal-fin aqua) (fourth increment-list))
 (move-incremental (tail-fin aqua) (fifth increment-list)))

(defmethod move-incremental ((aqua dolphin) increment-list)
 (move-incremental (body aqua) (first increment-list))
 (move-incremental (right-fin aqua) '(0))
 (move-incremental (left-fin aqua) '(0))
 (move-incremental (dorsal-fin aqua) '(0))
 (move-incremental (tail-fin aqua) '(0)))

(defun new-dolphin ()
 (setf flip (make-instance 'dolphin))
 (initialize flip))
```

```
; ** fin-link.cl **

(defclass link0 (rotary-link)
        ((twist-angle :initform 0)
         (link-length :initform 0.0)
         (inboard-joint-angle :initform 0)
         (inboard-joint-displacement :initform 0)
         (min-joint-angle :initform (deg-to-rad -360))
         (max-joint-angle :initform (deg-to-rad 360))
         (node-list :initform '((0 0 0 1) (0 0 0 1) (-20 0 0 1)))
         (polygon-list :initform '((1 2)))))

(defclass link1 (rotary-link)
        ((twist-angle :initform 0.0)
         (link-length :initform 0.0)
         (inboard-joint-angle :initform 0)
         (inboard-joint-displacement :initform 0)
         (min-joint-angle :initform 0)
         (max-joint-angle :initform 0)
         (node-list
          :initform '((0 0 0 1)
                      (60 22.5 -60 1) (60 22.5 -75 1) (39 22.5 -75 1)
                      (39 22.5 -60 1) ;right side
                      (60 -22.5 -60 1) (60 -22.5 -75 1) (39 -22.5 -75 1)
                      (39 -22.5 -60 1) ;left side
                      (39 22.5 -60 1) (39 22.5 -75 1) (39 -22.5 -75 1)
                      (39 -22.5 -60 1) ;back
                      (60 22.5 -60 1) (60 22.5 -75 1) (60 -22.5 -75 1)
                      (60 -22.5 -60 1))) ;front
         (polygon-list :initform  '((1 2 3 4) (5 6 7 8) (9 10 11 12)
                                    (13 14 15 16)))))

(defclass link2 (rotary-link)
        ((twist-angle :initform (deg-to-rad 90))
         (link-length :initform 0)
         (inboard-joint-angle :initform (deg-to-rad 0))
         (inboard-joint-displacement :initform 0)
         (min-joint-angle :initform (deg-to-rad -90))
         (max-joint-angle :initform (deg-to-rad 90))
         (node-list :initform '((0 0 0 1) (0 0 0 1) (0 0 0 1)))
         (polygon-list :initform '((1 2)))))
```

```
(defclass link3 (rotary-link)
        ((twist-angle :initform (deg-to-rad 0))
         (link-length :initform 0)
         (inboard-joint-angle :initform (deg-to-rad 0))
         (inboard-joint-displacement :initform 0)
         (min-joint-angle :initform (deg-to-rad -90))
         (max-joint-angle :initform (deg-to-rad 90))
         (node-list
          :initform '((0 0 0 1) (3 2 0 1) (-1 2 0 1) (-1 5 0 1)))
         (polygon-list
          :initform '((1 2 3)))))

(defmethod update-A-matrix ((link link))
 (with-slots (twist-angle link-length inboard-joint-angle
                          inboard-joint-displacement A-matrix) link
                          (setf A-matrix
                           (dh-matrix
                            (cos inboard-joint-angle)
                            (sin inboard-joint-angle) (cos twist-angle)
                            (sin twist-angle)
                            link-length inboard-joint-displacement))))

(defmethod rotate ((link rotary-link) angle)
 (setf (inboard-joint-angle link) angle)
 (update-A-matrix link)
 (setf (H-matrix link) (matrix-multiply (H-matrix (inboard-link link))
                                        (A-matrix link)))
 (transform-node-list link))

(defmethod rotate-link ((link rotary-link) angle)
 (cond ((> angle (max-joint-angle link))
        (rotate link (max-joint-angle link))
        (setf (motion-limit-flag link) t))
       ((< angle (min-joint-angle link))
        (rotate link (min-joint-angle link))
        (setf (motion-limit-flag link) t))
       (t (rotate link angle) (setf (motion-limit-flag link) nil))))
```

```
; ** fin.cl **

(defclass dolphin-fin ()
 ((fin-attachment-angle
   :initarg :fin-attachment-angle
   :accessor fin-attachment-angle)
  (link2
   :initform (make-instance 'link2)
   :accessor link2)
  (link3
   :initform (make-instance 'link3)
   :accessor link3)
  (motion-complete-flag
   :initform nil
   :accessor motion-complete-flag)
  (previous-fin-position
   :initform nil
   :accessor previous-fin-position)
  (current-fin-position
   :initform nil
   :accessor current-fin-position)))

(defclass right-fin (dolphin-fin)
 ((link2 :initform (make-instance 'link2
                  :link-length 22
                  :twist-angle (deg-to-rad 90)
                  :inboard-joint-displacement 1))
  (link3 :initform (make-instance 'link3
                  :node-list '((0 0 0 1) (0 10 10 1)
                          (44 30 0 1) (0 0 40 1))))))

(defclass left-fin (dolphin-fin)
 ((link2 :initform (make-instance 'link2
                  :link-length 22
                  :twist-angle (deg-to-rad 90)
                  :inboard-joint-displacement 1))
  (link3 :initform (make-instance 'link3
                  :node-list '((0 0 0 1) (0 10 -10 1)
                          (44 30 0 1) (0 0 -40 1))))))
```

```
(defclass dorsal-fin (dolphin-fin)
 ((link2 :initform (make-instance 'link2
                  :link-length 0
                  :twist-angle (deg-to-rad 90)))
  (link3 :initform (make-instance 'link3
                  :node-list '((0 0 0 1) (0 -22 -10 1)
                               (0 -55 -20 1) (0 -22 17 1))))))

(defclass tail-fin (dolphin-fin)
 ((link2 :initform (make-instance 'link2
                  :link-length -90
                  :twist-angle (deg-to-rad 90)
                  :node-list '((0 0 0 1) (0 0 0 1))))
  (link3 :initform (make-instance 'link3
                  :node-list '((0 0 0 1) (-22 0 22 1)
                               (-22 0 -22 1) (0 0 0 1))
                  :polygon-list '((1 2 3))))))

(defmethod initialize-fin ((fin dolphin-fin) (aqua dolphin))
 (setf (inboard-link (link2 fin)) (body aqua))
 (setf (inboard-link (link3 fin)) (link2 fin))
 (rotate-link (link2 fin) (fin-attachment-angle fin))
 (rotate-link (link3 fin) (inboard-joint-angle (link3 fin)))
 (setf (current-fin-position fin)
     (firstn 3 (first (transform-node-list (link3 fin))))))

(defmethod take-picture ((camera camera) (fin dolphin-fin))
 (take-picture camera (link3 fin)))

(defmethod move-incremental ((fin dolphin-fin) increment-list)
 (rotate-link (link2 fin) (fin-attachment-angle fin))
 (rotate-link (link3 fin)
          (+ (first increment-list) (inboard-joint-angle
                                    (link3 fin))))
 (setf (previous-fin-position fin) (current-fin-position fin))
 (setf (current-fin-position fin)
     (firstn 3 (first (transformed-node-list (link3 fin)))))
 (setf (motion-complete-flag fin) (not (or (motion-limit-flag
                                    (link3 fin))))))
```

```
; ** hunter.cl **

(defclass hunter ()
        ((dolphin
          :initform (make-instance 'dolphin)
          :accessor dolphin)
         (sans
          :initform (make-instance 'sans)
          :accessor sans)
         (node-list
          :initform '((0 0 0 1))
          :accessor node-list)))

(defmethod initialize ((hunt1 hunter))
  (setf delta-t (get-delta-t (sans hunt1)))
  (initialize (dolphin hunt1))
  (initialize-sans (sans hunt1) hunt1)
  (setf (node-list hunt1) (append (node-list (dolphin hunt1))
                                  (node-list (link1 (sans hunt1)))))
  (update-velocity-growth-rate hunt1)
  (transform-node-list hunt1))

(defmethod make-hunter-picture ()
  (setf hunt1 (make-instance 'hunter))
  (initialize hunt1)
  (x-picture)
  (setf cam1 (make-instance 'camera))
  (take-picture cam1 hunt1))

(defun create-hunter-1 ()
  (setf hunt1 (make-instance 'hunter))
  (initialize hunt1))

(defmethod take-picture ((camera camera) (hunt1 hunter))
  (take-picture camera (dolphin hunt1))
  (take-picture camera (link1 (sans hunt1))))

(defun hunter-picture ()
  (create-hunter-1)
  (create-camera-1)
  (take-picture camera-1 hunt1))
```

```
(defmethod move-incremental ((hunt1 hunter) increment-list)
 (move-incremental (dolphin hunt1) increment-list)
 (move-incremental (sans hunt1) (first increment-list)))

(defmethod move-and-flap-incremental ((hunt1 hunter) increment-list)
 (move-and-flap-incremental (dolphin hunt1) increment-list)
 (move-incremental (sans hunt1) (first increment-list)))

(defmethod update-posture ((hunt1 hunter) delta-t)
 (setf increment-list (list
                (list
                 (* delta-t (sixth (velocity (sans hunt1))))
                 (* delta-t (fifth (velocity (sans hunt1))))
                 (* delta-t (fourth (velocity (sans hunt1))))
                 (* delta-t (first (velocity (sans hunt1))))
                 (* delta-t (second (velocity (sans hunt1))))
                 (* delta-t (third (velocity (sans hunt1)))))))
 (move-incremental hunt1 increment-list))

(defmethod update-posture-2 ((hunt1 hunter))
 (setf increment-list (list
                (list 0 0 0
                    (delta-range (sans hunt1))
                    0
                    (delta-depth (sans hunt1)))))
 (move-incremental hunt1 increment-list))

(defmethod x-picture ()
 (setf x (make-instance 'rigid-body
       :node-list '((0 0 0 1) (50 0 50 1) (-50 0 -50 1)
                (50 0 -50 1) (-50 0 50 1))))
 (initialize x))

(defun hunter-video ()
 (create-hunter-1)
 (create-video-camera-1)
 (take-picture camera-1 hunt1))

(defun hunter-movie ()
 (create-hunter-1)
 (create-movie-camera-1)
 (take-picture camera-1 hunt1))
```

```
(defun hunter-3-movies ()
 (create-hunter-1)
 (create-3-movie-cameras)
 (take-three))

(defmethod new-hunter ()
 (setf hunt1 (make-instance 'hunter))
 (initialize hunt1))

(defmethod new-picture ((camera camera) (hunt1 hunter))
 (erase-camera-window camera)
 (take-picture camera hunt1))

(defmethod display-acceleration-profile ((hunt1 hunter)
                                         (camera camera) acceleration-profile)
 (dolist (element acceleration-profile)
  (read-acceleration-profile (sans hunt1) element)
  (setf (location (body (dolphin hunt1))) (location (sans hunt1)))
  (setf (velocity (body (dolphin hunt1))) (velocity (sans hunt1)))
  (setf (velocity-growth-rate (body (dolphin hunt1)))
   (velocity-growth-rate (sans hunt1)))
  (update-posture hunt1 delta-t)
  (take-picture camera hunt1)))

(defmethod read-acceleration-profile ((body rigid-body) element)
 (setf (first (velocity-growth-rate body))
  (* 100 (first element)))          ;longitudinal
 (setf (turn-radius (sans hunt1)) (* 100 (second element)))
 (setf delta-t (- (fourth element) (third element)))
 (calculate-velocity-and-normal body delta-t)
 (setf (climb-angle (sans hunt1))
  (rad-to-deg (atan (/ (third (velocity-growth-rate body))
                       (first (velocity-growth-rate body))))))
 (update-velocity body delta-t)
 (update-location (sans hunt1) delta-t)
 (update-delta-range (sans hunt1))
 (set-accelerometer hunt1))
```

```
(defmethod calculate-velocity-and-normal ((body rigid-body) delta-t)
 (setf delta-v (* (first (velocity-growth-rate  body)) delta-t))
 (setf new-velocity (+ delta-v (first (velocity body))))
 (setf (third (velocity-growth-rate body))
  (/ (* new-velocity new-velocity) (turn-radius (sans hunt1)))))

(defmethod set-accelerometer ((hunt1 hunter))
 (setf (sensed-acceleration (sans hunt1))
  (+ (first (velocity-growth-rate (sans hunt1)))
    (* *gravity* (sin (deg-to-rad (climb-angle (sans hunt1)))))))
 (setf apparent-climb-angle
  (rad-to-deg (asin (+ (sin (deg-to-rad (climb-angle (sans hunt1))))
                (/ (first (velocity-growth-rate
                      (sans hunt1))) *gravity*)))))
 (update-horizontal-error hunt1))

(defmethod update-horizontal-error ((hunt1 hunter))
 (setf (horizontal-error (sans hunt1))
  (* (delta-depth (sans hunt1))
    (- (/ 1 (tan (deg-to-rad (climb-angle (sans hunt1)))))
      (/ 1 (tan (deg-to-rad apparent-climb-angle)))))))

(defmethod display-acceleration-profile-2 ((hunt1 hunter)
                                (camera camera) acceleration-profile)  (dolist (element acceler-
ation-profile)
  (read-acceleration-profile-2 (sans hunt1) element)
  (setf (location (body (dolphin hunt1))) (location (sans hunt1)))
  (setf (velocity (body (dolphin hunt1))) (velocity (sans hunt1)))
  (setf (velocity-growth-rate (body (dolphin hunt1)))
   (velocity-growth-rate (sans hunt1)))
  (update-posture-2 hunt1)
  (take-picture camera hunt1)
  (move-horizontal x (+ (horizontal-error (sans hunt1))
                (first (location (sans hunt1))))
            (second (location (sans hunt1)))
            (third (location (sans hunt1))))
  (take-picture camera x)))
```

```
(defmethod read-acceleration-profile-2 ((body rigid-body) element)
 (setf (first (velocity-growth-rate body))
  (* 100 (first element)))          ;longitudinal
 (setf (turn-radius (sans hunt1)) (* 100 (second element)))
 (setf delta-t (- (fourth element) (third element)))
 (calculate-velocity-and-normal-2 body delta-t)
 (update-delta-range (sans hunt1))
 (update-velocity body delta-t)
 (update-location-2 (sans hunt1) delta-t)
 (set-accelerometer hunt1))

(defmethod calculate-velocity-and-normal-2 ((body rigid-body) delta-t)
 (setf delta-v (* (first (velocity-growth-rate  body)) delta-t))
 (setf new-velocity (+ delta-v (first (velocity body))))
 (setf (third (velocity-growth-rate body))
  (* (first (velocity-growth-rate body))
    (tan (deg-to-rad (climb-angle (sans hunt1)))))))

(defmethod update-velocity-growth-rate ((hunt1 hunter))
 (update-velocity-growth-rate (body (dolphin hunt1)))
 (update-velocity-growth-rate (sans hunt1)))

(defmethod transform-node-list ((hunt1 hunter))
 (transform-node-list (sans hunt1))
 (transform-node-list (body (dolphin hunt1))))

(defmethod run-profile ((hunt1 hunter))
 (display-acceleration-profile hunt1 cam1 profile)
 (move-horizontal x (+ (horizontal-error (sans hunt1))
                (first (location (sans hunt1))))
            (second (location (sans hunt1)))
            (third (location (sans hunt1))))
 (take-picture cam1 x))

(defun h-error ()
 (/ (horizontal-error (sans hunt1)) 100))
```

```
(defun camera-setup ()
 (setf display (make-instance 'camera))
 (setf (camera-window display) (cw:make-window-stream
                    :borders 5
                    :left 800
                    :bottom 800
                    :width 1400
                    :height 600
                    :title "Dolphin in Motion"
                    :activate-p t))
 (move-camera display (deg-to-rad -90) 0 0 2800 6000 -2800) ; side view
 (zoom-camera display 20))
; (move-camera display 0 0 0 -40000 0 0)          ; rear view
; (move-camera display 0 (deg-to-rad -90) 0 0 0 -40000)  ; top view

; Calculates the error from a given climb angle
(defun calculate-error (climb-angle profile display)
 (load "hunter.cl")
 (setf hunt1 (make-instance 'hunter))
 (initialize hunt1)
 (setf (location (sans hunt1)) '(0 0 0))
; (three-cameras)
; (take-three)
 (x-picture)
 (take-picture display hunt1)
 (setf (climb-angle (sans hunt1)) climb-angle)
 (setf total-distance '0)
 (display-acceleration-profile-2 hunt1 display profile)
 (move-horizontal x (+ (horizontal-error (sans hunt1))
                (first (location (sans hunt1))))
            (second (location (sans hunt1)))
            (third (location (sans hunt1)))))
 (take-picture display hunt1)
 (take-picture display x)
 (h-error))

(defmethod calculate-error-demo (climb-angle profile)
 (camera-setup)
 (calculate-error (climb-angle profile display)))
```

```
; Calculates the error by determining climb angle from the given turn radius
(defmethod error-given-turn-radius ()
 (setf hunt1 (make-instance 'hunter))
 (initialize hunt1)
; (three-cameras)
; (take-three)
 (setf display (make-instance 'camera))
 (setf (camera-window display) (cw:make-window-stream
                    :borders 5
                    :left 600
                    :bottom 600
                    :width 600
                    :height 600
                    :title "Dolphin in Motion"
                    :activate-p t))
 (move-camera display (deg-to-rad -90) 0 0 0 10000 0) ; side view
; (move-camera display 0 0 0 -40000 0 0)          ; rear view
; (move-camera display 0 (deg-to-rad -90) 0 0 0 -40000)  ; top view
 (x-picture)
 (take-picture display hunt1)
 (display-acceleration-profile hunt1 display profile)
 (move-horizontal x (+ (horizontal-error (sans hunt1))
              (first (location (sans hunt1))))
           (second (location (sans hunt1)))
           (third (location (sans hunt1))))
 (take-picture display hunt1)
 (take-picture display x))
```

```
; ** link.cl **

(defclass link (rigid-body)
 ((motion-limit-flag
   :initform nil
   :accessor motion-limit-flag)
  (twist-angle
   :initarg :twist-angle
   :accessor twist-angle)
  (link-length
   :initarg :link-length
   :accessor link-length)
  (inboard-joint-angle
   :initarg :inboard-joint-angle
   :accessor inboard-joint-angle)
  (inboard-joint-displacement
   :initarg :inboard-joint-displacement
   :accessor inboard-joint-displacement)
  (inboard-link
   :initarg :inboard-link
   :accessor inboard-link)
  (A-matrix
   :accessor A-matrix)))

(defclass rotary-link (link)
 ((min-joint-angle
   :initarg :min-joint-angle
   :accessor min-joint-angle)
  (max-joint-angle
   :initarg :max-joint-angle
   :accessor max-joint-angle)))

(defclass sliding-link (link)
 ((min-joint-displacement
   :initarg :min-joint-displacement
   :accessor min-joint-displacement)
  (max-joint-displacement
   :initarg :max-joint-displacement
   :accessor max-joint-displacement)))
```

```
; ** load-files-3.cl **

(defun load-hunter ()

; Abstract class files
  (load "rigid-body.cl")
  (load "link.cl")

; Specific Graphics and Display files
  (load "camera.cl")

; Specific Hunter files
  (load "profiles.cl")
  (load "robot-kinematics.cl")
  (load "dolphin.cl")
  (load "hunter.cl")
  (load "sans.cl")
  (load "fin.cl")
  (load "fin-link.cl"))

(defun compile-hunter ()
; Specific Graphics and Display files
  (compile-file "camera.cl")   ; must be first to avoid name conflicts
                               ; may complain it can't finddeg-to-rad()
; Abstract class files
  (compile-file "rigid-body.cl")
  (compile-file "link.cl")

; Specific Hunter files
  (compile-file "profiles.cl")
  (compile-file "robot-kinematics.cl")
  (compile-file "dolphin.cl")
  (compile-file "hunter.cl")
  (compile-file "sans.cl")
  (compile-file "fin.cl")
  (compile-file "fin-link.cl"))

(defun load-compiled-hunter ()
; Specific Graphics and Display files
  (load "camera.fasl") ; needs to be first to avoid name conflicts error
```

```
; Abstract class files
 (load "rigid-body.fasl")
 (load "link.fasl")

; Specific Hunter files
 (load "profiles.fasl")
 (load "robot-kinematics.fasl")
 (load "dolphin.fasl")
 (load "hunter.fasl")
 (load "sans.fasl")
 (load "fin.fasl")
 (load "fin-link.fasl"))
```

```
; ** load-files.cl **

(defun load-hunter ()

; Abstract class files
  (load "rigid-body.cl")
  (load "link.cl")

; Specific Graphics and Display files
  (load "camera.cl")
; Specific Hunter files
  (load "profiles.cl")
  (load "robot-kinematics.cl")
  (load "dolphin.cl")
  (load "hunter.cl")
  (load "sans.cl")
  (load "fin.cl")
  (load "fin-link.cl"))

(defun compile-hunter ()
; Specific Graphics and Display files
  (compile-file "camera.cl")   ; must be first to avoid name conflicts
                               ; may complain it can't finddeg-to-rad()
; Abstract class files
  (compile-file "rigid-body.cl")
  (compile-file "link.cl")

; Specific Hunter files
  (compile-file "profiles.cl")
  (compile-file "robot-kinematics.cl")
  (compile-file "dolphin.cl")
  (compile-file "hunter.cl")
  (compile-file "sans.cl")
  (compile-file "fin.cl")
  (compile-file "fin-link.cl"))

(defun load-compiled-hunter ()
; Specific Graphics and Display files
  (load "camera.fasl") ; needs to be first to avoid name conflicts error
```

```
; Abstract class files
 (load "rigid-body.fasl")
 (load "link.fasl")

; Specific Hunter files
 (load "profiles.fasl")
 (load "robot-kinematics.fasl")
 (load "dolphin.fasl")
 (load "hunter.fasl")
 (load "sans.fasl")
 (load "fin.fasl")
 (load "fin-link.fasl"))
```

```
; ** newsetup.cl **

(load "load-files.cl")
(load-hunter)
(compile-hunter)
(load-compiled-hunter)


; ** setup-3.cl **

(load "load-files-3.cl")
(load-hunter)
(compile-hunter)
(load-compiled-hunter)


; ** setup.cl **

(load "load-files.cl")
(load-compiled-hunter)
```

```
; ** rigid-body.cl **

(defclass rigid-body
  ()
 ((location          ;The three-vector (x y z) in world coordinates.
  :initform '(0 0 0)
  :initarg :location
  :accessor location)
 (velocity           ;The six-vector (u v w p q r) in body coordinates.
  :initform '(1 1 1 .1 .1 .1)
  :initarg :velocity
  :accessor velocity)
 (velocity-growth-rate  ;The vector (u-dot v-dot w-dot p-dot q-dot r-dot).
  :initform '(0 0 0 0 0 0)
  :accessor velocity-growth-rate)
 (forces-and-torques   ;The vector (Fx Fy Fz L M N) in body coordinates.
  :initform '(0 0 0 0 0 0)
  :initarg :forces-and-torques
  :accessor forces-and-torques)
 (moments-of-inertia   ;The vector (Ix Iy Iz) in principal axis coordinates.
  :initform '(1 1 1)
  :initarg :moments-of-inertia
  :accessor moments-of-inertia)
 (mass
  :initform 1
  :initarg :mass
  :accessor mass)
 (node-list   ;(x y z 1) in body coord for each node. Starts with (0 0 0 1).
  :initform '((0 0 0 1))
  :initarg :node-list  ; x
  :accessor node-list)
 (polygon-list
  :initform '((1 2) (3 4))
  :initarg :polygon-list
  :accessor polygon-list)
 (transformed-node-list  ;(x y z 1) in earth coord for each node in node-list
  :accessor transformed-node-list)
 (H-matrix
  :initform (unit-matrix 4)
  :accessor H-matrix)
 (current-time
  :initform 1979487650
  :accessor current-time)
```

```
   (delta-depth
    :initform 0
    :accessor delta-depth)
   (delta-range
    :initform 0
    :accessor delta-range)
   (depth
    :initform 0
    :accessor depth)
   (range
    :initform 0
    :accessor range)))

(defmethod initialize ((body rigid-body))
 (update-velocity-growth-rate body)
 (transform-node-list body))

(defmethod move-horizontal ((body rigid-body) x y z)
 (setf (H-matrix body)
  (homogeneous-transform 0 0 0 x y z))
 (transform-node-list body)
 (setf (location body) (firstn 3 (first (transformed-node-list body)))))

(defmethod move-body ((body rigid-body) azimuth elevation roll x y z)
 (setf (H-matrix body)
  (homogeneous-transform azimuth elevation roll x y z))
 (transform-node-list body)
 (update-position body))


(defmethod move-incremental ((body rigid-body) increment-list)
 (setf (H-matrix body)
  (matrix-multiply (H-matrix body)
            (homogeneous-transform
             (first increment-list) ;body z rotation
             (second increment-list) ;body y rotation
             (third increment-list) ;body x rotation
             (fourth increment-list) ;body x translation
             (fifth increment-list) ;body y translation
             (sixth increment-list)))) ;body z translation
 (transform-node-list body)
(update-position body))
```

```
(defmethod get-delta-t ((body rigid-body))
 (let* ((new-time (get-internal-real-time))
        (delta-t (/ (- new-time (current-time body)) 1000000)))
  (setf (current-time body) new-time)
  delta-t))

(defmethod start-timer ((body rigid-body))
 (setf (current-time body) (get-internal-real-time)))

(defmethod update-velocity-growth-rate ((body rigid-body))
 (setf (velocity-growth-rate body)   ;Assumes principal axis coordinates with   (multiple-
value-bind              ;origin at center of gravity of body.
     (Fx Fy Fz L M N   u v w p q r   Ix Iy Iz) ;Declares local variables.
     (values-list              ;Values assigned.
      (append
       (forces-and-torques body) (velocity body) (moments-of-inertia body)))
   (list (+ (* v r) (* -1 w q) (/ Fx (mass body))
            (* *gravity* (first (third (H-matrix body)))))
         (+ (* w p) (* -1 u r) (/ Fy (mass body))
           (* *gravity* (second (third (H-matrix body)))))
         (+ (* u q) (* -1 v p) (/ Fz (mass body))
           (* *gravity* (third (third (H-matrix body)))))
         (/ (+ (* (- Iy Iz) q r) L) Ix)
         (/ (+ (* (- Iz Ix) r p) M) Iy)
         (/ (+ (* (- Ix Iy) p q) N) Iz))))) ;Value returned.

(defmethod update-velocity ((body rigid-body) delta-t)(defmethod update-posture ((body
rigid-body) delta-t) ;Euler integration.
 (move-incremental body
            (list (* delta-t (sixth (velocity body)))
                  (* delta-t (fifth (velocity body)))
                  (* delta-t (fourth (velocity body)))
                  (* delta-t (first (velocity body)))
                  (* delta-t (second (velocity body)))
                  (* delta-t (third (velocity body))))))

(defmethod update-location ((body rigid-body) delta-t)
 (setf (location body)
  (vector-add (location body)
        (scalar-multiply delta-t (velocity body)))))
```

```
(defmethod update-location-2 ((body rigid-body) delta-t)
  (setf (first (location body)) (range body))
  (setf (third (location body)) (depth body)))

(defmethod transform-node-list ((body rigid-body))
  (setf (transformed-node-list body)
    (mapcar #'(lambda (node-location)
          (post-multiply (H-matrix body) node-location))
        (node-list body))))

(defmethod update-position ((body rigid-body))
  (setf (location body) (firstn 3 (first (transformed-node-list body)))))

(defmethod update-delta-range ((body rigid-body))
  (setf delta-distance (+ (* (first (velocity-growth-rate body)) delta-t delta-t .5)
                (* (first (velocity body)) delta-t)))
  (setf total-distance (+ total-distance delta-distance))
  (setf (depth body)
    (* (sin (deg-to-rad (climb-angle body))) total-distance))
  (setf (delta-depth body)
    (* (sin (deg-to-rad (climb-angle body))) delta-distance))
  (setf (range body)
    (* (cos (deg-to-rad (climb-angle body))) total-distance))
  (setf (delta-range body)
    (* (cos (deg-to-rad (climb-angle body))) delta-distance)))

(defconstant *gravity* 981)      ;cm/sec

(defun deg-to-rad (angle) (* .017453292519943295 angle))

(defun rad-to-deg (radian) (/ radian .017453292519943295))

(defun feet-to-cm (feet) (* 30.48 feet))

(defun cm-to-feet (cm) (/ cm 30.48))


  (setf (velocity body)
    (vector-add (velocity body)
          (scalar-multiply delta-t (velocity-growth-rate body)))))
```

```
; ** robot-kinematics.cl **

(defun transpose (matrix)          ;A matrix is a list of row vectors.
 (cond ((null (cdr matrix)) (mapcar 'list (car matrix)))
       (t (mapcar 'cons (car matrix) (transpose (cdr matrix))))))

(defun dot-product (vector-1 vector-2)  ;A vector is a list of numerical atoms
 (apply '+ (mapcar '* vector-1 vector-2)))

(defun vector-magnitude (vector) (sqrt (dot-product vector vector)))

(defun post-multiply (matrix vector)
 (cond ((null (rest matrix)) (list (dot-product (first matrix) vector)))
       (t (cons (dot-product (first matrix) vector)
                (post-multiply (rest matrix) vector)))))

(defun pre-multiply (vector matrix)
 (post-multiply (transpose matrix) vector))

(defun matrix-multiply (A B)          ;A and B are conformable matrices.
 (cond ((null (cdr A)) (list (pre-multiply (car A) B)))
       (t (cons (pre-multiply (car A) B) (matrix-multiply (cdr A) B)))))

;L is a list of names of conformable matrices.
(defun chain-multiply (L)
 (cond ((null (cddr L)) (matrix-multiply (eval (car L)) (eval (cadr L))))
       (t (matrix-multiply (eval (car L)) (chain-multiply (cdr L))))))

(defun cycle-left (matrix) (mapcar 'row-cycle-left matrix))

(defun row-cycle-left (row) (append (cdr row) (list (car row))))

(defun cycle-up (matrix) (append (cdr matrix) (list (car matrix))))

(defun unit-vector (one-column length)  ;Column count starts at 1.
 (do ((n length (1- n))
      (vector nil (cons (cond ((= one-column n) 1) (t 0)) vector)))
     ((zerop n) vector)))
```

```
(defun unit-matrix (size)
  (do ((row-number size (1- row-number))
       (I nil (cons (unit-vector row-number size) I)))
      ((zerop row-number) I)))

;A and B are matrices with equal number of rows.
(defun concat-matrix (A B)
  (cond ((null A) B)
        (t (cons (append (car A) (car B)) (concat-matrix (cdr A) (cdr B))))))

(defun augment (matrix)
  (concat-matrix matrix (unit-matrix (length matrix))))

(defun normalize-row (row) (scalar-multiply (/ 1.0 (car row)) row))

(defun scalar-multiply (scalar vector)
  (cond ((null vector) nil)
        (t (cons (* scalar (car vector))
                 (scalar-multiply scalar (cdr vector))))))

;Reduces first column to (1 0 ... 0).
(defun solve-first-column (matrix)
  (do* ((remaining-row-list matrix (rest remaining-row-list))
        (first-row (normalize-row (first matrix)))
        (answer (list first-row)
                (cons (vector-add
                        (first remaining-row-list)
                        (scalar-multiply
                         (- (caar remaining-row-list)) first-row)) answer)))
       ((null (rest remaining-row-list)) (reverse answer))))

(defun vector-add (vector-1 vector-2) (mapcar '+ vector-1 vector-2))

(defun vector-subtract (vector-1 vector-2) (mapcar '- vector-1 vector-2))

;Returns leftmost square matrix from argument.
(defun first-square (matrix)
  (do ((size (length matrix))
       (remainder matrix (rest remainder))
       (answer nil (cons (firstn size (first remainder)) answer)))
      ((null remainder) (reverse answer))))
```

```
(defun firstn (n list)
 (cond ((zerop n) nil)
     (t (cons (first list) (firstn (1- n) (rest list))))))

(defun max-car-firstn (n list)
 (append (max-car-first (firstn n list)) (nthcdr n list)))

(defun matrix-inverse (M)
 (do ((M1 (max-car-first (augment M))
       (cond ((null M1) nil)
           (t (max-car-firstn n (cycle-left (cycle-up M1))))))
    (n (1- (length M)) (1- n)))
   ((or (minusp n) (null M1)) (cond ((null M1) nil) (t (first-square M1))))
  (setq M1 (cond ((zerop (caar M1)) nil) (t (solve-first-column M1))))))

;L is a list of lists. This function finds list with
;largest car and moves it to head of list of lists.
(defun max-car-first (L)
 (cond ((null (cdr L)) L)
     (t (if (> (abs (caar L)) (abs (caar (max-car-first (cdr L))))) L
        (append (max-car-first (cdr L)) (list (car L)))))))

(defun dh-matrix (cosrotate sinrotate costwist sintwist length translate)
 (list (list cosrotate (- (* costwist sinrotate))
         (* sintwist sinrotate) (* length cosrotate))
     (list sinrotate (* costwist cosrotate)
         (- (* sintwist cosrotate)) (* length sinrotate))
     (list 0. sintwist costwist translate) (list 0. 0. 0. 1.)))

(defun homogeneous-transform (azimuth elevation roll x y z)
 (rotation-and-translation (sin azimuth) (cos azimuth) (sin elevation)
                 (cos elevation) (sin roll) (cos roll) x y z))

(defun rotation-and-translation (spsi cpsi sth cth sphi cphi x y z)
 (list (list (* cpsi cth) (- (* cpsi sth sphi) (* spsi cphi))
         (+ (* cpsi sth cphi) (* spsi sphi)) x)
     (list (* spsi cth) (+ (* cpsi cphi) (* spsi sth sphi)
                                 (- (* spsi sth cphi) (* cpsi sphi)) y)
     (list (- sth) (* cth sphi) (* cth cphi) z)
     (list 0. 0. 0. 1.)))
```

```
;H is a 4x4 homogeneous transformation matrix.
(defun inverse-H (H)
 (let* ((minus-P (list (- (fourth (first H)))
                       (- (fourth (second H)))
                       (- (fourth (third H)))))
        (inverse-R (transpose (first-square (reverse (rest (reverse H))))))
        (inverse-P (post-multiply inverse-R minus-P)))
   (append (concat-matrix inverse-R (transpose (list inverse-P)))
           (list (list 0 0 0 1)))))

(setf x '((1 2) (3 4)))
)
```

```
; ** sans.cl **

(defclass sans (rigid-body)
 ((location
  :initform '(0 0 0))     ;x y z (rigid body dolphin in a plane)
 (velocity
  :initform '(0 0 0 0 0 0))
 (velocity-growth-rate
  :initform '(0 0 0 0 0 0))     ;tangential and normal
 (forces-and-torques
  :initform '(0 0 0 0 0 0))
 (moments-of-inertia
  :initform '(1 1 1))
 (mass
  :initform 5)
 (H-matrix
  :initform (homogeneous-transform 0 0 0 0 0 0))
 (sensed-acceleration
  :initform '(0 0 0)
  :initarg :sensed-acceleration
  :accessor sensed-acceleration)
 (turn-radius
  :initform 0
  :initarg :turn-radius
  :accessor turn-radius)
 (climb-angle
  :initform 0
  :accessor climb-angle)
 (delta-depth
  :initform 0
  :accessor delta-depth)
 (horizontal-error
  :initform 0
  :accessor horizontal-error)
 (fin-attachment-angle
  :initarg :fin-attachment-angle
  :initform 0
  :accessor fin-attachment-angle)
 (link0
  :initform (make-instance 'link0)
  :accessor link0)
 (link1
  :initform (make-instance 'link1     ;in centimeters
```

```
        :node-list '((0 0 0 1)
                (25 10 -22 1) (37 10 -22 1)
                (37 10 -30 1) (25 10 -30 1) ;right side
                (25 -10 -22 1) (37 -10 -22 1)
                (37 -10 -30 1) (25 -10 -30 1) ;left side
                (37 10 -22 1) (37 -10 -22 1)
                (37 -10 -30 1) (37 10 -30 1) ;front
                (25 10 -22 1) (25 -10 -22 1)
                (25 -10 -30 1) (25 10 -30 1) ;back

                (25 3 -30 1) (25 -3 -30 1)
                (25 -3 -33 1) (25 3 -33 1) ; back of antenna
                (31 3 -30 1) (31 -3 -30 1)
                (31 -3 -33 1) (31 3 -33 1) ; front of antenna
                (25 3 -30 1) (31 3 -30 1)
                (31 3 -33 1) (25 3 -33 1) ; right side
                (25 -3 -30 1) (31 -3 -30 1)
                (31 -3 -33 1) (25 -3 -33 1) ; left side

                (25 2 -33 1) (31 2 -33 1) (28 0 -44 1)
                (31 2 -33 1) (31 -2 -33 1) (28 0 -44 1)
                (31 -2 -33 1) (25 -2 -33 1) (28 0 -44 1)
                (25 -2 -33 1) (25 2 -33 1) (28 0 -44 1)) ; top of antenna

        :polygon-list '((1 2 3 4) (5 6 7 8) (9 10 11 12)
                (13 14 15 16) (17 18 19 20)
                (21 22 23 24) (25 26 27 28)
                (29 30 31 32) (33 34 35) (36 37 38)
                (39 40 41) (42 43 44))

        :min-joint-angle '0
        :max-joint-angle '0)
 :accessor link1)
(motion-complete-flag
 :initform nil
 :accessor motion-complete-flag)
(previous-fin-position
 :initform nil
 :accessor previous-fin-position)
(current-fin-position
 :initform nil
 :accessor current-fin-position)))
```

```
(defmethod initialize-sans ((sans sans) (hunt1 hunter))
 (setf (inboard-link (link0 sans)) (body (dolphin hunt1)))
 (setf (inboard-link (link1 sans)) (link0 sans))
 (rotate-link (link0 sans) (fin-attachment-angle sans))
 (rotate-link (link1 sans) (inboard-joint-angle (link1 sans)))
 (setf (current-fin-position sans)
  (firstn 3 (first (transformed-node-list (link1 sans)))))
 (update-velocity-growth-rate sans)
 (transform-node-list (link1 sans)))

(defmethod move-incremental ((sans sans) increment-list)
 (rotate-link (link0 sans) (fin-attachment-angle sans))
 (rotate-link (link1 sans)
        (+ (first increment-list) (inboard-joint-angle (link1 sans))))
 (setf (previous-fin-position sans) (current-fin-position sans))
 (setf (current-fin-position sans)
  (firstn 3 (first (transformed-node-list (link1 sans)))))
 (setf (motion-complete-flag sans) (not (or (motion-limit-flag (link0 sans))
                                            (motion-limit-flag (link1 sans))))))
```

APPENDIX C: PLOTS OF STATIC-TEST DATA

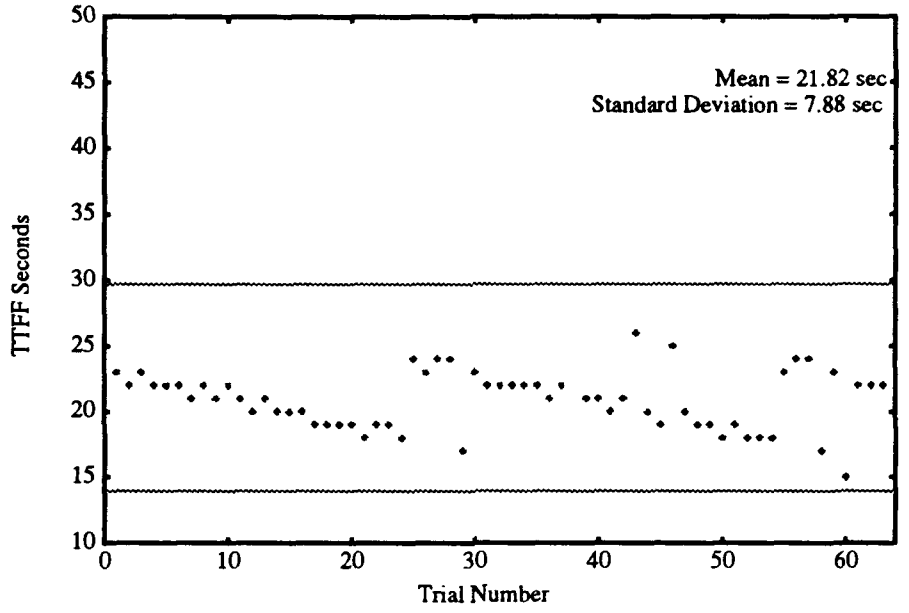


Figure 31. Motorola First Acquisition Time After Ninety Seconds Off

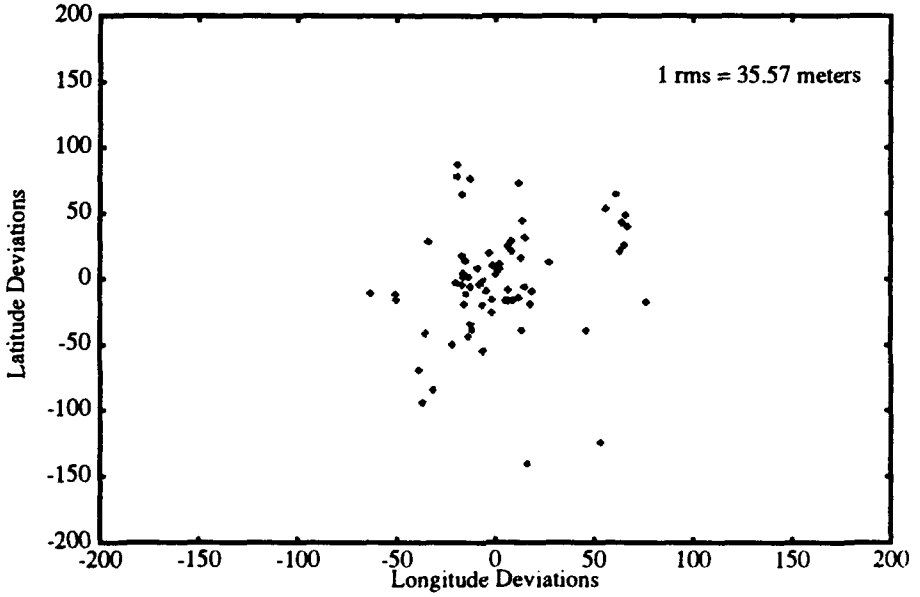


Figure 32. Motorola First Acquisition Accuracy After Ninety Seconds Off

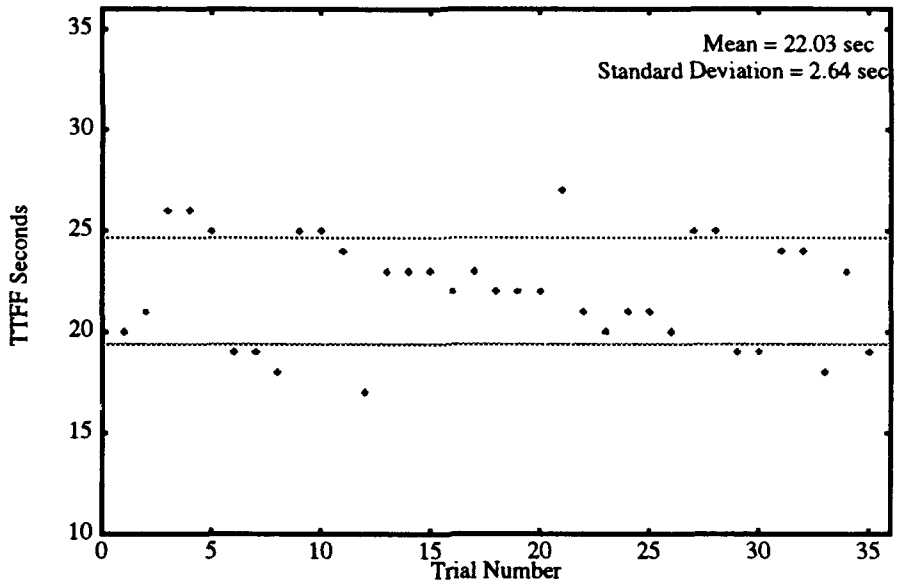


Figure 33. Motorola First Acquisition Time After Ten Minutes Off

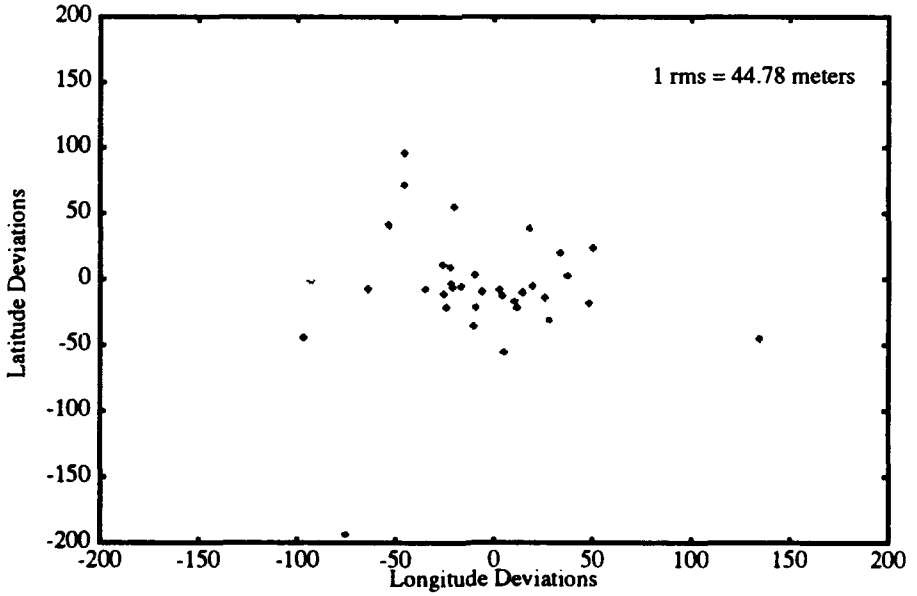


Figure 34. Motorola First Acquisition Accuracy After Ten Minutes Off

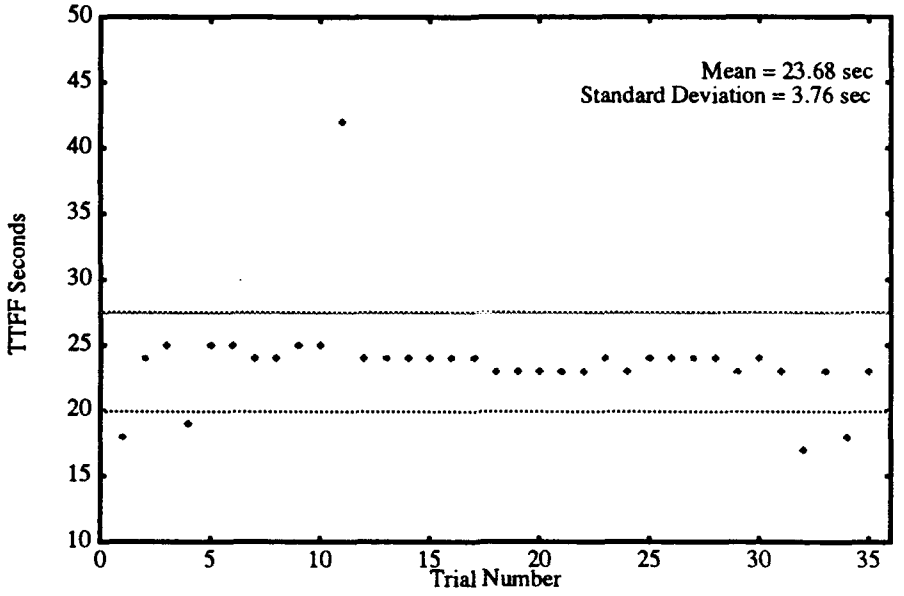


Figure 35. Motorola First Acquisition Time After 30 Minutes Off

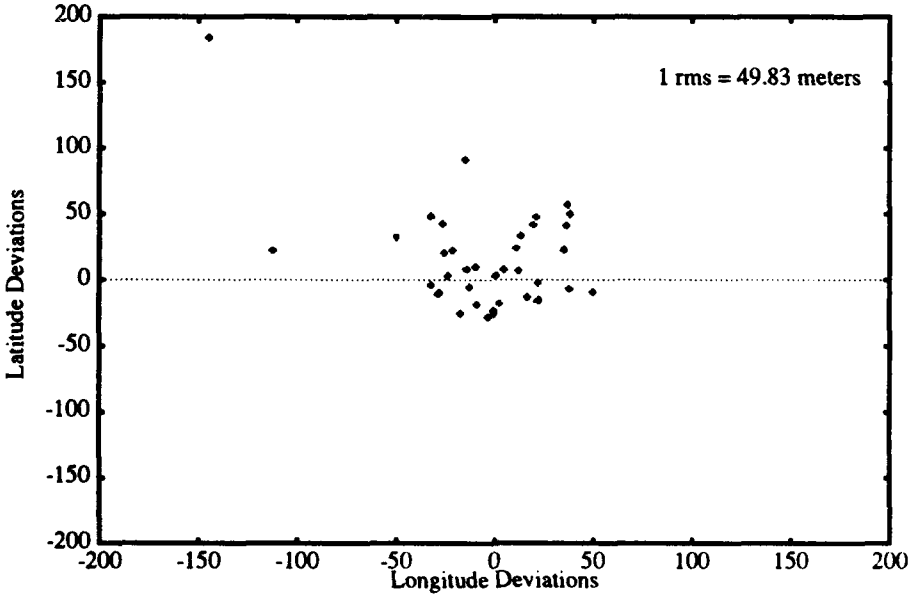


Figure 36. Motorola First Acquisition Accuracy After 30 Minutes Off

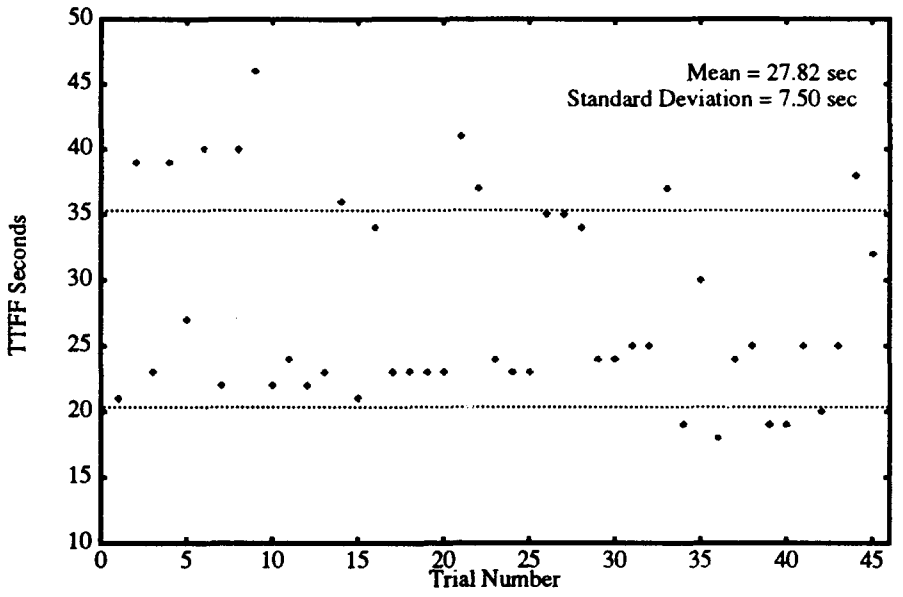


Figure 37. Motorola First Acquisition Time After 1 Hour Off

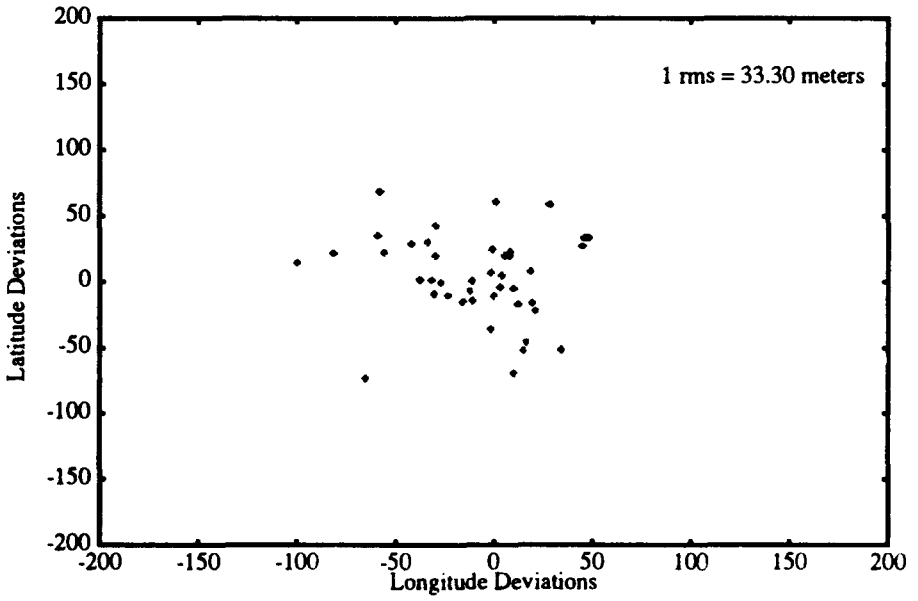


Figure 38. Motorola First Acquisition Accuracy After 1 Hour Off

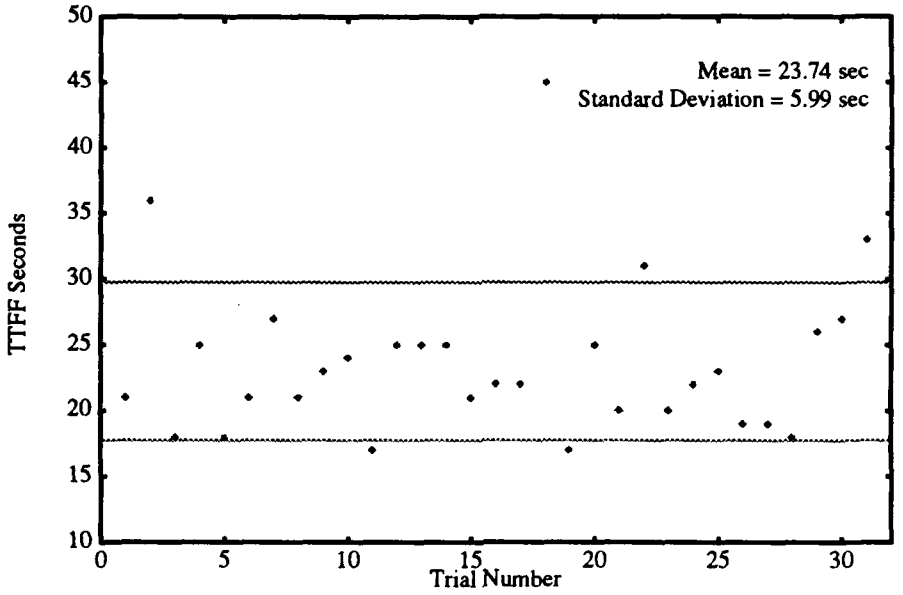


Figure 39. Motorola First Acquisition Time After 3 Hours Off

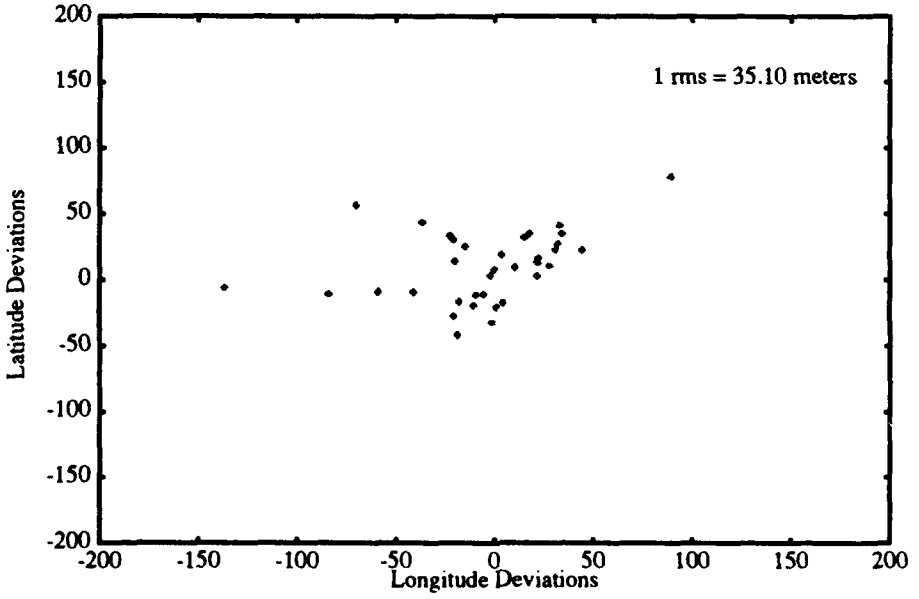


Figure 40. Motorola First Acquisition Accuracy After 3 Hours Off

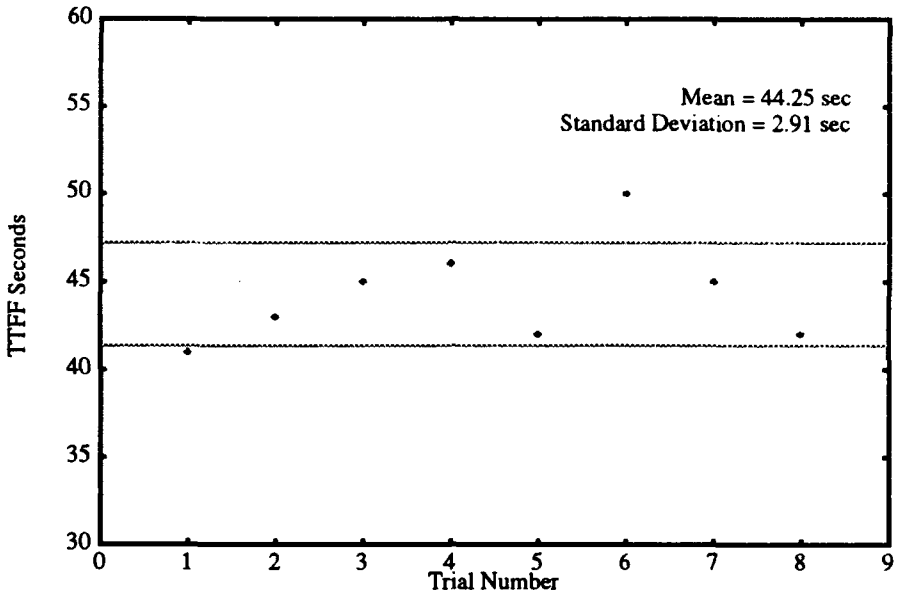


Figure 41. Motorola First Acquisition Time After 6 Hours Off

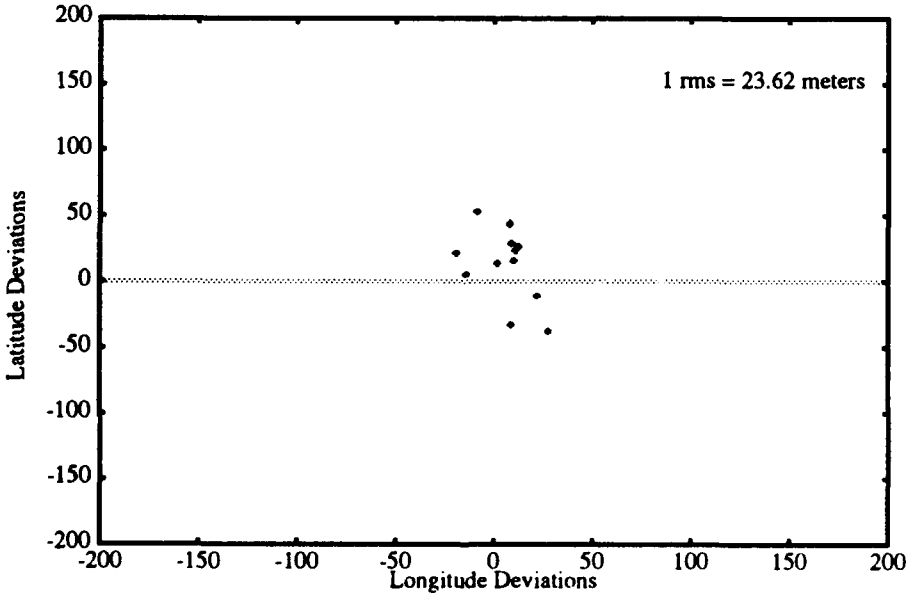


Figure 42. Motorola First Acquisition Accuracy After 6 Hours Off

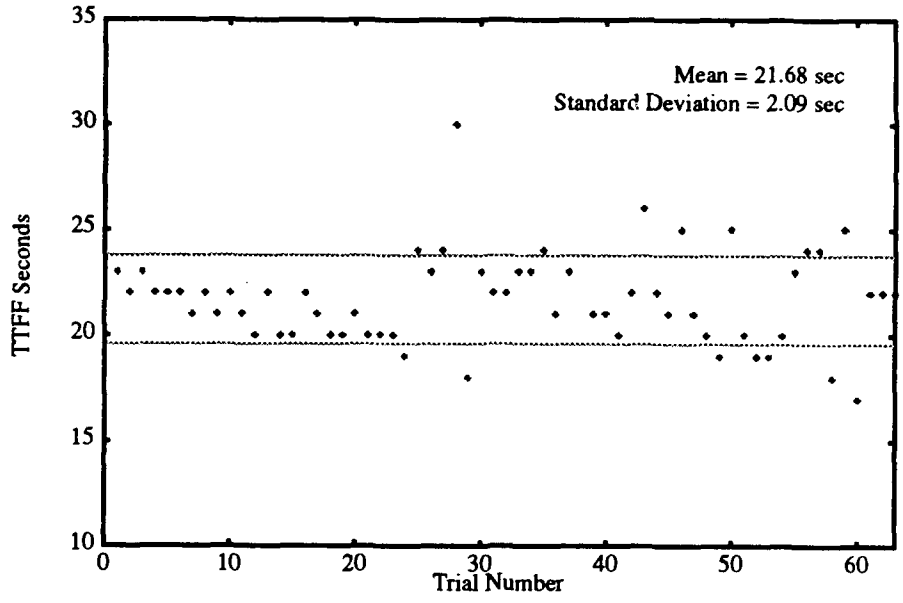


Figure 43. Motorola Time to First Fix After 90 Seconds Off

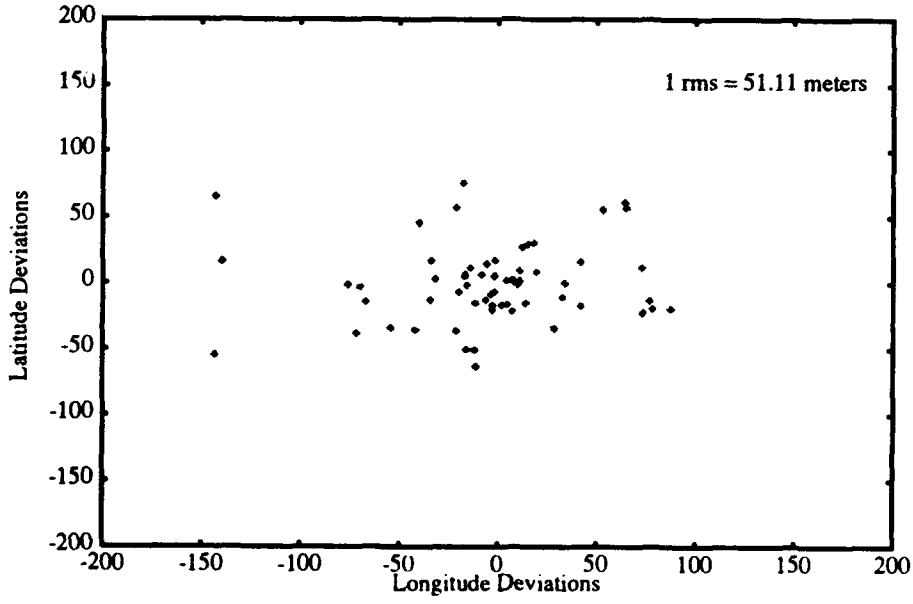


Figure 44. Motorola First Fix Accuracy After 90 Seconds Off

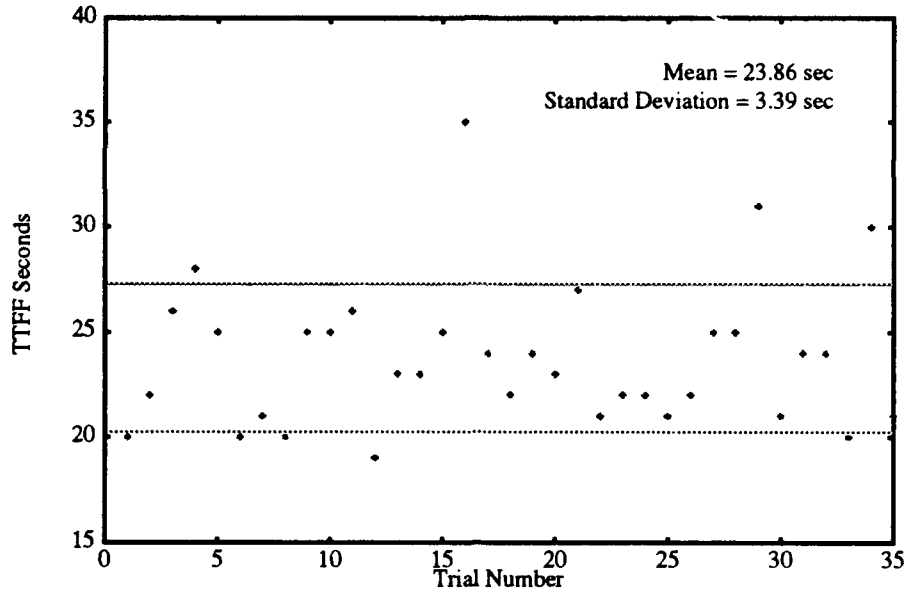


Figure 45. Motorola Time to First Fix After 10 Minutes Off

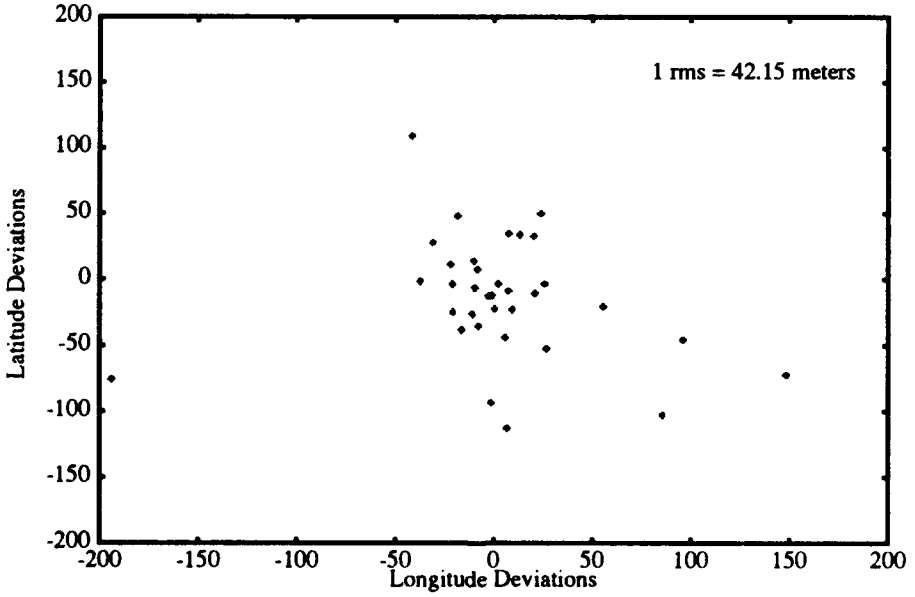


Figure 46. Motorola First Fix Accuracy After 10 Minutes Off

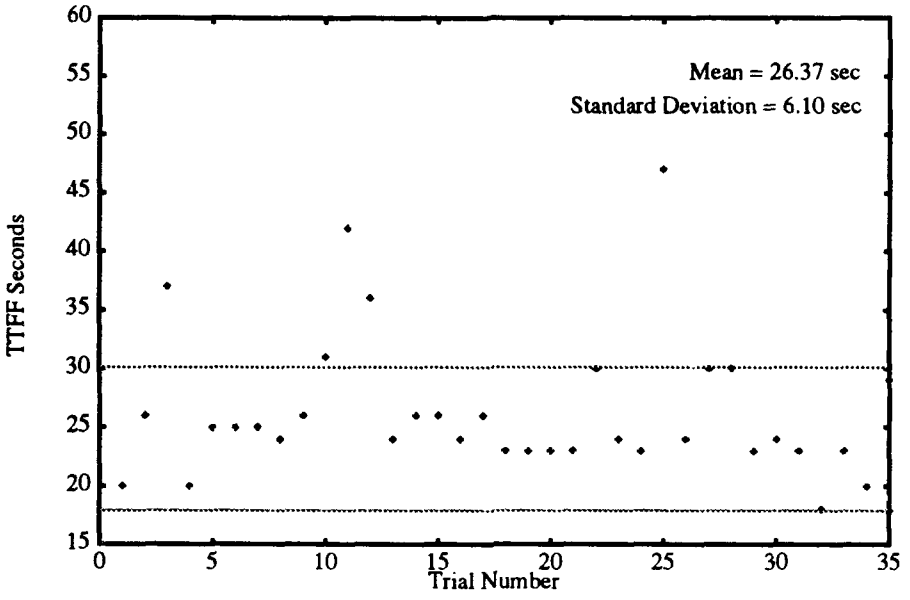


Figure 47. Motorola Time to First Fix After 30 Minutes Off

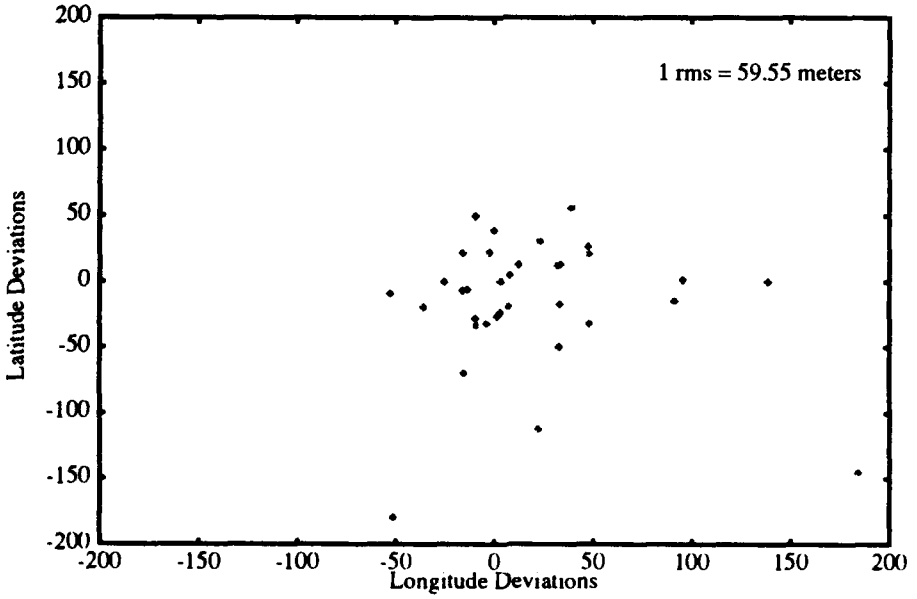


Figure 48. Motorola First Fix Accuracy After 30 Minutes Off

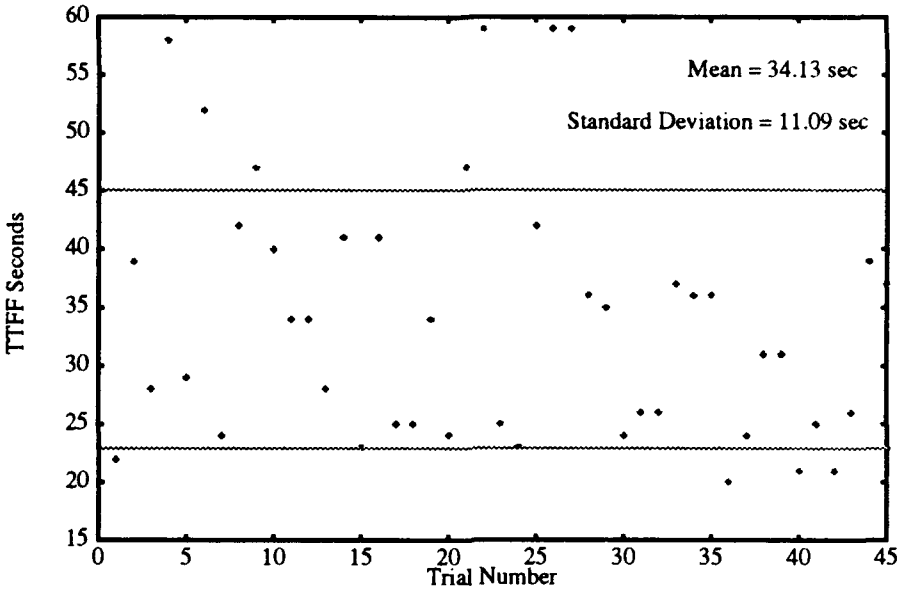


Figure 49. Motorola Time to First Fix After 1 Hour Off

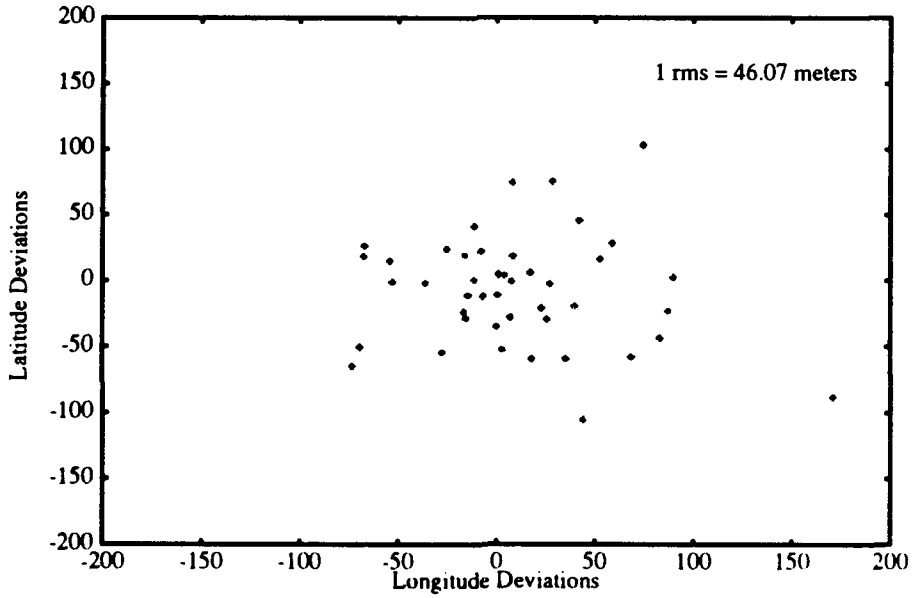


Figure 50. Motorola First Fix Accuracy After 1 Hour Off

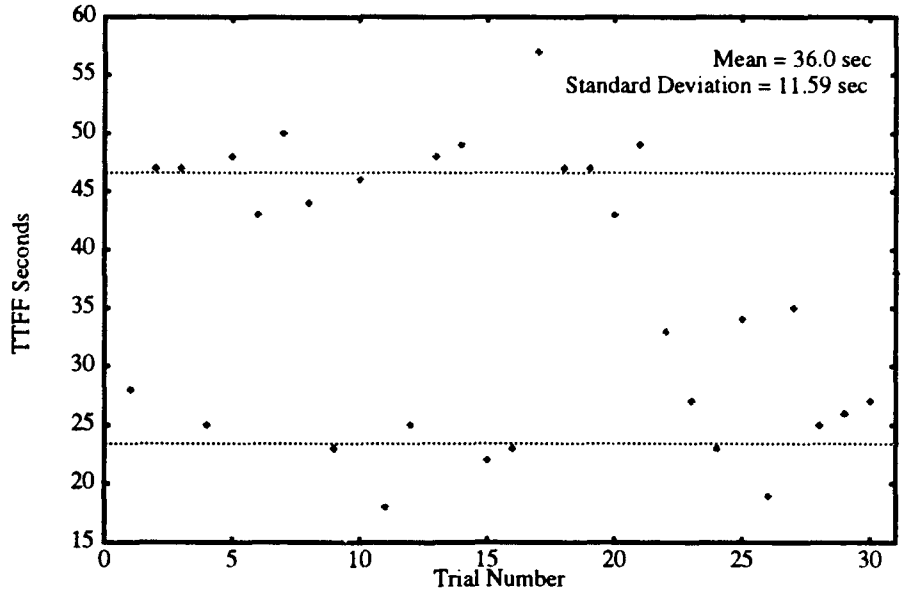


Figure 51. Motorola Time to First Fix After 3 Hours Off

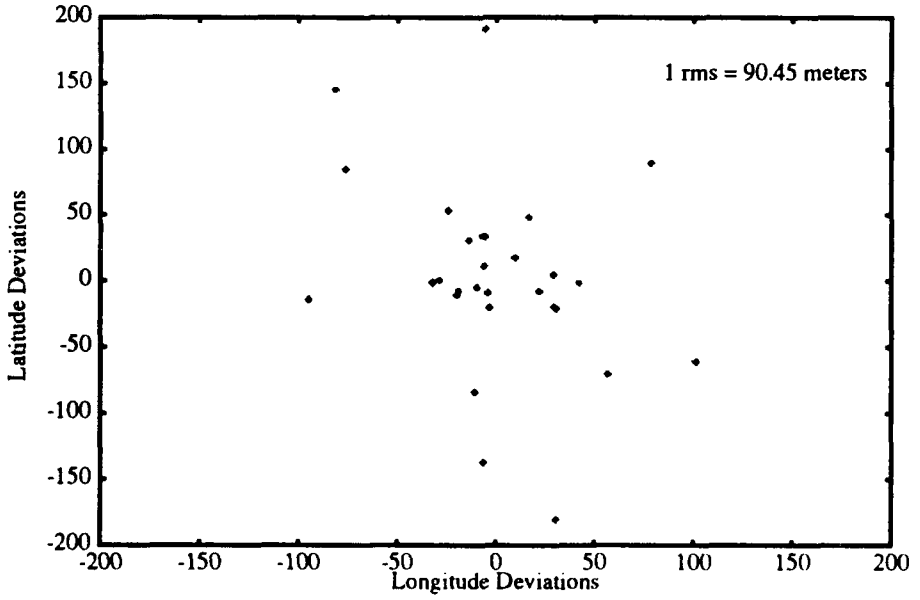


Figure 52. Motorola First Fix Accuracy After 3 Hours Off

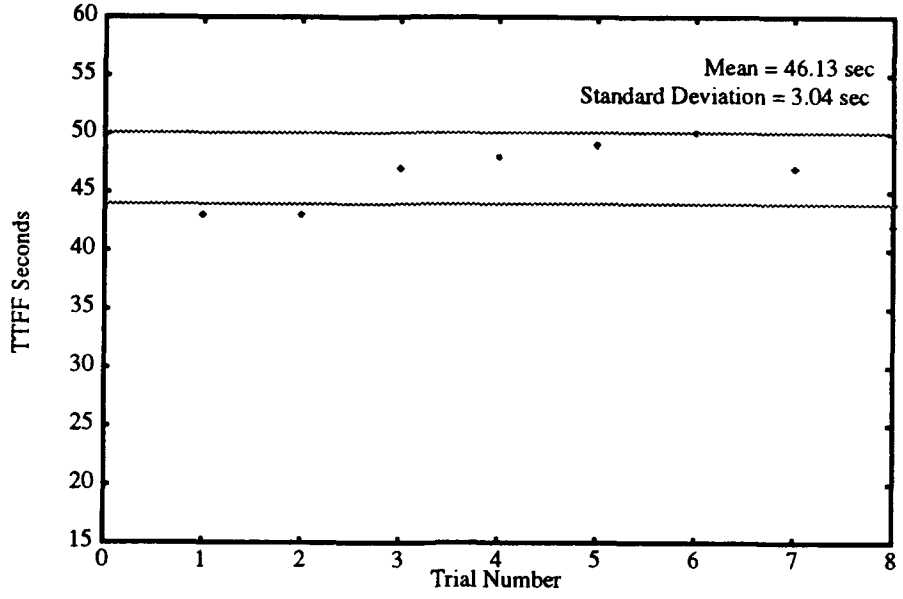


Figure 53. Motorola Time to First Fix After 6 Hours Off

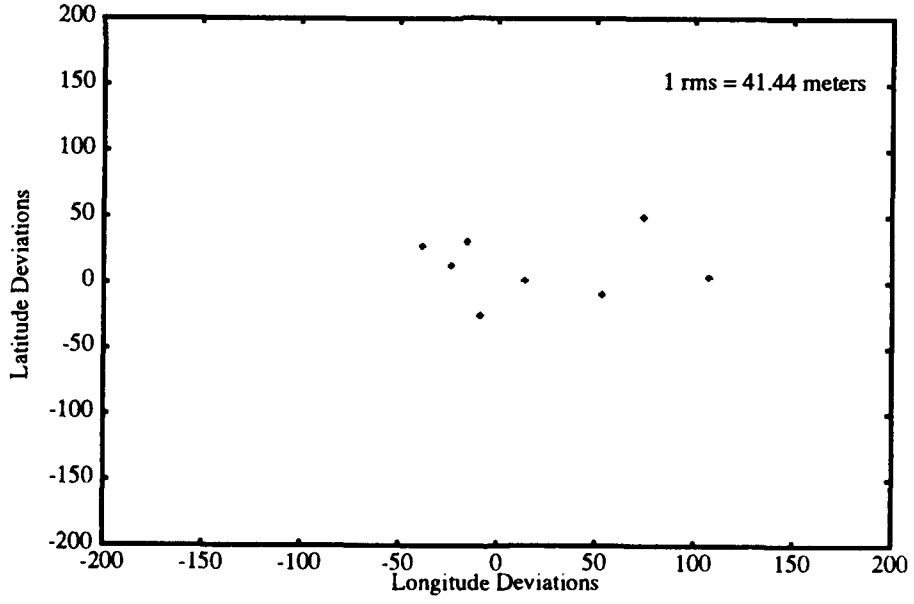


Figure 54. Motorola First Fix Accuracy After 6 Hours Off

APPENDIX D: GLOSSARY OF TERMS

Almanac — Data transmitted by a GPS satellite which includes orbit information on all the satellites, clock correction, and atmospheric delay parameters. These data are used to facilitate rapid satellite acquisition. The orbit information is a subset of the ephemeris data with reduced accuracy.

Azimuth — A horizontal direction expressed as the angular distance between a fixed direction, such as North, and the direction of the object.

Baseline — The three-dimensional (3D) vector distance between a pair of stations for which simultaneous GPS data has been collected and processed with differential techniques.

C/A Code — The Coarse/Acquisition (or Clear/Acquisition) code modulated onto the GPS L1 signal. This code is a sequence of 1023 pseudorandom binary biphase modulations on the GPS carrier at a chipping rate of 1.023 MHz, thus having a code repetition period of one millisecond. This code was selected to provide good acquisition properties.

CLOS — Common LISP Object System

Differential Processing — GPS measurements can be differenced between receiver, satellites, and epochs. Although many combinations are possible, the present convention for differential processing of GPS phase measurements, is to take differences between receivers (single difference), then between satellite (double difference), then between measurement epochs (triple difference). A single-difference measurement between receivers is the instantaneous difference in phase of the signal from the same satellite, measured by two receivers simultaneously. A double-difference measurement is obtained by differencing the single difference for one satellite with respect to the corresponding single difference for a chosen reference satellite. A triple-difference measurement is the difference between a double difference at one epoch of time and the same double difference at the previous epoch of time.

Differential (relative) Processing — Determination of relative coordinates of two or more receivers which are simultaneously tracking the same satellites. Dynamic differential positioning is a real-time calibration technique achieved by sending corrections to the roving user from one or more monitor stations. Static differential GPS involves determining baseline vectors between pairs of receivers.

Dilution of Precision	A description of the purely geometrical contribution to the uncertainty in a position fix, given by the expression DOP = SQRT TRACE (A A) where A A is the design matrix for the instantaneous position solution (dependent on satellite-receiver geometry). The DOP factor depends on the parameters of the position-fix solution. Standard terms for the GPS application are: GDOP: Geometric (three position coordinates plus clock offset in the solution) PDOP: Position (three coordinates) GDOP: Horizontal (two horizontal coordinates) VDOP: Vertical (height only) TDOP: Time (clock offset only) RDOP: Relative (normalized to 60 seconds)
Doppler Shift	The apparent change in frequency of a received signal due to the rate of change of the range between the transmitter and receiver.
Elevation	Height above mean sea level. Vertical distance above the geoid.
Ephemeris	A list of accurate positions or locations of a celestial object as a function of time. Available as "broadcast ephemeris" or as post-processed "precise ephemeris".
Epoch	Measurement interval or data frequency, as in making observations every 15 seconds. Loading data using 30-second epochs, means loading every other measurement.
EEPROM	Electrically Erasable Programmable Read Only Memory, a random access memory device that is generally read but can be erased and rewritten electrically, often without removing it from its application system.
EPROM	Erasable Programmable Read Only Memory, a random access device in which individual words can be read but not written. An EPROM can be erased, often by irradiating it with ultraviolet light.
GDOP	Geometric Dilution of Precision. The relationship between errors in user position and time and in satellite range. $GDOP^2 = PDOP^2 + TDOP^2$
GMT	Greenwich Mean Time. Local solar mean time at Greenwich Meridian.
Inclination	The angle between the orbital plane of a body and some reference plane (e.g. equatorial plane).

INS	Inertial Navigation System, which contains an Inertial Measurement Unit (IMU).
Kalman Filter	A numerical method used to track a time-varying signal in the presence of noise. If the signal can be characterized by some number of parameters that vary slowly with time, then Kalman filtering can be used to tell how incoming raw measurements should be processed to best estimate those parameters as a function of time.
Kinematic Surveying	A form of continuous differential carrier-phase surveying requiring only short periods of data observations. Operational constraints include starting from or determining a known baseline, and tracking a minimum of four satellites. One receiver is statically located at a control point, while others are moved between points to be measured.
L1	The primary L-band signal radiated by each NAVSTAR satellite at 1575.42 MHz. The L1 beacon is modulated with the C/A and P codes, and with the NAV message. L2 is centered at 1227.60 MHz and is modulated with the P code and the NAV message.
L Band	The radio frequency band extending from 390 MHz to (nominally) 1550 MHz.
Multichannel Receiver	A receiver containing many independent channels. Such a receiver offers highest signal-to-noise ratio (SNR) because each channel tracks one satellite continuously.
Multipath	Interference similar to "ghosts" on a television screen which occurs when GPS signals arrive at an antenna having traversed different paths. The signal traversing the longer path will yield a larger pseudorange estimate and increase the error. Multiple paths may arise from reflections from structures near the antenna.
Multipath Error	A positioning error resulting from interference between radio waves which have traveled between the transmitter and the receiver by two paths of different electrical lengths.
Multiplexing Channel	A receiver channel which is sequenced through several satellite signals (each from a specific satellite and at a specific frequency) at a rate which is synchronous with the satellite message bit-rate (50 bits per second, or 20 milliseconds per bit). Thus, one complete sequence is completed in a multiple of 20 milliseconds.

NAVDATA	The 1500-bit Navigation Message broadcast by each satellite at 50 bps on both L1 or L2 beacons. This message contains system time, clock correction parameters, ionospheric delay model parameters, and the vehicle's ephemeris and health. This information is used to process GPS signals to obtain user position and velocity.
NAVSTAR	The name given to GPS satellites, built by Rockwell International, which is an acronym formed from NAViation System with Time and Ranging.
PCMCIA	PC Memory Card International Association, a voluntary association responsible for defining the electrical interface to and physical requirements of memory and I/O modules that are approximately the size of a standard credit card.
PCB	Printed Circuit Board
P Code	The protected or precise code used on both L1 and L2 GPS beacons. This code will be made available by the DoD only to authorized users. The P code is a very long (about 10^{14} bits) sequence of pseudorandom binary biphase modulations on the GPS carrier at a chipping rate of 10.23 MHz which does not repeat itself for about 38 weeks. Each satellite uses a one-week segment of this code which is unique to each GPS satellite, and is reset each week.
PDOP	Position Dilution of Precision, a unitless figure of merit expressing the relationship between the error in user position and the error in satellite position. Geometrically, PDOP is proportional to 1 divided by the volume of the pyramid formed by lines running from the receiver to four satellites observed. Values considered good for positioning are small, such as 3. Values greater than 7 are considered good for positioning are small, such as 3. Values greater than 7 are considered poor. Thus, small PDOP is associated with widely separated satellites. A small value of PDOP is important in positioning, but much less so in surveying.
PLCC	Plastic Leadless Chip Carrier, a high-density, integrated circuit package designed to fit in a socket for easy installation and removal.
Point Positioning	A geographic position produced from one receiver in a stand-alone mode. At best, position accuracy obtained from a stand-alone receiver is 15 - 25 meters, depending on the geometry of the satellites.

PPS	Precise Positioning Service. The highest level of military dynamic positioning accuracy that will be provided by GPS, based on the dual frequency P code and having high anti-jam and anti-spoof qualities.
PRN	Pseudorandom noise, a sequence of digital 1s and 0s which appear to be randomly distributed like noise, but which can be exactly reproduced. The important property of PRN codes is that they have a low autocorrelation value for all delays or lags except when they are exactly coincident. Each NAVSTAR satellite has its own unique C/A and P pseudorandom-noise codes.
Pseudorange	A measure of the apparent propagation time from the satellite to the receiver antenna, expressed as a distance. Pseudorange is obtained by multiplying the apparent signal-propagation time by the speed of light. Pseudorange differs from the actual range by the amount that the satellite and user clocks are offset, by propagation delays, and other errors. The apparent propagation time is determined from the time shift required to align (correlate) a replica of the GPS code generated in the receiver with the received GPS code. The time shift is the difference between the time of signal reception (measured in the receiver time frame) and the time of emission (measured in the satellite time frame).
RAM	Random Access Memory, a memory that can conveniently read from and write to any location. Random access memory contrasts with sequential access memory which can access only one location without repositioning a read/write head. Although there are other random-access devices (such as an EPROM), the acronym RAM refers only to devices that can read and write a single location at a time.
ROM	Read Only Memory, a random access memory that can be read but not written. Data in a ROM is either built into the ROM when it is manufactured or is written into the ROM in a once-only operation called programming. Devices that can be programmed are more properly called PROMs.
RDOP	Relative Dilution of Precision. See Dilution of Precision.
RMS	Root Mean Square.

SA	Selective Availability. A DoD program to control the accuracy of pseudorange measurements, whereby the user receives a false pseudorange which is in error by a controlled amount. Differential GPS techniques can reduce these effects for local applications.
SEP	Spherical Error Probable, a statistical measure of precision defined as the 50th percentile value of the three-dimensional position error statistics. Thus, half of the results are within a 3D DEP value.
Sidereal Day	Time between two successive upper transits of the vernal equinox.
Spread Spectrum	The received GPS signal is a wide bandwidth, low-power signal (-160 dBW). This property results from modulating the L-band signal with a PRN code in order to spread the signal energy over a bandwidth which is much greater than the signal information bandwidth. This is done to provide the ability to receive all satellites unambiguously and to provide some resistance to noise and multipath.
SPS	Standard Positioning Service, using the C/A code to provide a minimum level of dynamic or static positioning capability. The accuracy of this service will be set at a level consistent with national security.
TDOP	Time Dilution of Precision. See Dilution of Precision.
TOW	Time of Week, in seconds from midnight Sunday UTC.
UTC	Universal Time. Local solar mean time at Greenwich Meridian.
VDOP	Vertical Dilution of Precision. See Dilution of Precision.

LIST OF REFERENCES

[BELL92] Bellingham, J.G., Consi, T.R., Tedrow, U., and DiMassa, D., "Hyperbolic Acoustic Navigation for Underwater Vehicles: Implementation and Demonstration", *Proceedings of the 1992 Symposium on Autonomous Underwater Vehicle Technology*, IEEE Oceanic Engineering Society, June 2-3, 1992, Washington, D.C., pp. 304-309.

[BELL93-1] Bellingham, J.G., Bales, J.W., Atwood, D.K., Perrier, M., Goudey, C.A., Consi, T.R. and Chryssostomidis, C., "Performance Characteristics of the Odyssey AUV", *Proceedings of the 8th International Symposium on Unmanned Untethered Submersible Technology*, September 27-29, 1993, pp. 37-49.

[BELL93-2] Bellingham, J.G., Deffenbaugh, M., Leonard, J.J., Catipovic, J. and Schmidt, H., "Arctic Under-Ice Survey Operations", *Proceedings of the 8th International Symposium on Unmanned Untethered Submersible Technology*, September 27-29, 1993, pp. 50-59.

[BERG93] Bergem, O., "A Multibeam Sonar Based Positioning System for an AUV", *Proceedings of the 8th International Symposium on Unmanned Untethered Submersible Technology*, September 27-29, 1993, pp. 291-299.

[BOSS85] Bossler, J. D. Rear Admiral, Challstrom, C. W., "Global Positioning System Instrumentation and Federal Policy", *Proceedings of the First International Symposium on Precise Positioning with the Global Positioning System*, April 1985, pp. 1-9.

[BROW92-1] Brown, R.A., Cox, R.F., and Ebner, R.E., "Global Positioning Inertial Navigation System Development", *Proceedings of The Institute of Navigation GPS-92*, Albuquerque, NM, September 16-18, 1992, pp. 697-705.

[BROW92-2] Brown, G.B., *An Introduction to Random Signals and Applied Kalman Filtering*, pp. 437-441, John Wiley and Sons, Inc., New York, New York, 1992.

[BROW93] Brown, P., and Kirby-Smith, T., "Disposable GPS - Test Results of a Low Cost Sensor for Sonobuoy Applications", *Proceedings of The Institute of Navigation GPS-93*, Salt Lake City, UT, September 22-24, 1993, pp. 1417-1424.

[BUTL93] Butler, B. and den Hertog, V., "Theseus: A Cable-Laying AUV", *Proceedings of the 8th International Symposium on Unmanned Untethered Submersible Technology*, September 27-29, 1993, pp. 1-6.

[BYRN93] Byrnes, R.B., Nelson, M.L., McGhee, R.B., Kwak, S.H., and Healey, A.J., "Rational Behavior Model: An Implemented Tri-Level Multilingual Software Architecture for Control of Autonomous Underwater Vehicles", *Proceedings of the Eight International Symposium on Unmanned Untethered Submersible Technology*, September 27-29, 1993, Durham, NH, pp. 160-178.

[CLYN92] Clynch, J.R., Thurmond, G., Rosenfeld, L., and Schramm, R., "Error Characteristics of GPS Differential Positions and Velocities", *Proceedings of The Institute of Navigation GPS-92*, Albuquerque, NM, September 16 - 18, 1992, pp. 969.

[CLYN92-2] Clynch, J.R., "George 4.3 Documentation", Naval Postgraduate School, Monterey, CA, unpublished software input/output handler, July 1992.

[COAT93] Coates, R., and Wang, L., "The "WISP" Within-Structure Positioning System", *Proceedings of the 8th International Symposium on Unmanned Untethered Submersible Technology*, September 27-29, 1993, pp. 231-238.

[COCO90] Coco, D.S., Coker, C., and Clynch, J.R., "Mitigation of Ionospheric Effects for Single Frequency GPS Users", *Proceedings of The Institute of Navigation GPS-90*, Colorado Springs, CO, September 19-21, 1990, pp. 169-174.

[COX94] Cox, R. and Wei, S., "Advances in the State of the Art for AUV Inertial Sensors and Navigation Systems", *Proceedings of the 1994 Symposium on Autonomous Underwater Vehicle Technology*, July 19 - 20, 1994, Cambridge, MA, pp. 360-369.

[CRIS93] Cristi, R., "Sensor Based Navigation of an Autonomous Underwater Vehicle", *Proceedings of the 8th International Symposium on Unmanned Untethered Submersible Technology*, September 27-29, 1993, pp. 300-310.

[CZES93] Czeschin, M.E., Hyslop, G.L., Schieber, G.E., Schwartz, M.K., "Automated Mission Planning for the Standoff Land Attack Missile (SLAM)", *Proceedings of the Precision Strike Technology Symposium*, October 26-28, 1993, pp. 147-155.

[DAVI93] Davidson, S.L., *An Experimental Comparison of CLOS and C++ Implementations of an Object-Oriented Graphical Simulation of Walking Robot Kinematics*, M.S. Thesis, Naval Postgraduate School, Monterey, CA, March, 1993.

[ESP93-1] *E.S.P. 8680 Users Manual*, Dovatron International, Longmont, CO, February, 1993.*E.S.P. 8680 Users Manual*, Dovatron International, Longmont, CO, February, 1993.

[ESP93-2] *E.S.P. Analog to Digital Module Users Manual*, Dovatron International, Longmont, CO, February, 1993.

[ESP93-3] *E.S.P. Flash Disk Module Users Manual*, Dovatron International, Longmont, CO, February, 1993.

[FU87] Fu, K.S., Gonzalez, R.C., and Lee, C.S.G., *Robotics: Control, Sensing, Vision, and Intelligence*, McGraw-Hill, Inc., New York, 1987.

[GROS92] Grose, B.L., "The Application of the Correlation Sonar to Autonomous Underwater Vehicle Navigation", *Proceedings of the 1992 Symposium on Autonomous Underwater Vehicle Technology*, IEEE Oceanic Engineering Society, June 2-3, 1992, Washington, D.C., pp. 298-303.

[GYRO92] *Gyrochip Angular Rate Sensor Specifications*, Systron Donner Inertial Division, A BEI Electronics Co., Concord, CA, July, 1992.

[HADD93] Haddrell, A., and Meldrum, D., "A GPS Positioning System for an Automatic Submersible Research Vessel", *Proceedings of The Institute of Navigation GPS-93*, Salt Lake City, UT, September 22-24, 1993, pp. 701-709.

[HEAL92] Healey, A.J., McGhee, R.B., Cristi, R., Papoulias, F.A., Kwak, S.H., Kanayama, Y., Lee, Y., Shukla, S., Zaky, A., "Research on Autonomous Underwater Vehicles at the Naval Postgraduate School", *Naval Research Review*, Vol One, pp. 43-51.

[HEAL94] Healey, A.J., Marco, D.B., McGhee, R.B., Brutzman, D.P., Cristi, R., Papoulias, F.A., and Kwak, S.H., "Tactical/Execution Level Coordination for Hover Control of the NPS AUV II using Onboard Sonar Servoing", *Proceedings of the 1994 Symposium on Autonomous Underwater Vehicle Technology*, July 19 - 20, 1994, Cambridge, MA, pp. 129-138.

[HELL92] Hellard, M.J., "Magnetic Sensors for Unmanned Vehicles", *Journal of Unmanned Systems*", Spring 1992, pp. 49-51.

[HINR76] Hinrichs, P.R., "Advanced Terrain Correlation Techniques", *Record of the 1976 Position Location and Navigation Symposium*, IEEE Aerospace and Electronic Systems Society, 1976.

[JORG94] Jorgensen, K.V., Grose, B.L., and Crandall, F.A., "Doppler Precision Underwater Navigation", *Sea Technology*, Vol 35, No. 3, pp. 63-68.

[KOSC90] Koschmann, T.D., *The Common Lisp Companion*, John Wiley & Sons, New York, 1990.

[KRAU93] Krause, Reinhardt, "New GPS Modules Target Handhelds", *Electronic News*, Vol. 39, No. 1973, July 26, 1993, p. 4.

[KREM90] Kremer, G. T., Kalafus, R. M., Loomis, P. V. M., and Reynolds, J. C., "The Effect of Selective Availability on Differential Global Positioning System Corrections", *NAVIGATION, Journal of The Institute of Navigation*, Vol. 37 No. 1, Spring 1990.

[KVH91] "KVH C100 Multi-Purpose Digital Compass Sensor Preliminary Data Sheet", KVH Industries, Inc., Middletown, RI, 1991.

[KWAK93] Kwak, S.H., Stevens, C.D., Clynch, J.R., McGhee, R.B., and Whalen, R.H., "An Experimental Investigation of GPS/INS Integration for Small AUV Navigation", *Proceedings of the Eight International Symposium on Unmanned Untethered Submersible Technology*, September 27-29, 1993, Durham, NH, pp. 239-251.

[LEU93] Leu, C.T., Chao, J.J., and Lee, T.S., "GPS Based Underwater Positioning - A System Design", *Proceedings of The Institute of Navigation GPS-93*, Salt Lake City, UT, September 22-24, 1993, pp. 745-754.

[LIGH93] Light, R.D., "Miniature AUVs for Scientific Applications", *Sea Technology*, Vol. 34, No. 12, December 1993, pp.10-18.

[MCGH92] McGhee, R.B., Clynch, J.R., Kwak, S.H., and McKeon, J.B., *Technology Survey and Preliminary Design for Small AUV Navigation System*, Technical Report, NPSCS-92-001, Naval Postgraduate School, Monterey, CA, March, 1992.

[MCGH93] McGhee, R.B., "Finite Approximation of Climb Angle Using an Accelerometer", unpublished notes, Naval Postgraduate School, March 1993.

[MCKE92] McKeon, J.B., *Integration of GPS into a Small Underwater Navigation System*, M.S. Thesis, Naval Postgraduate School, Monterey, CA, March, 1992.

[MILL91] Miller, C., Application of Extended Kalman Filter to a Model-Based Navigator for an AUV, Master's Thesis, Naval Postgraduate School, Monterey, CA, December 1991.

[MOTO93-1] *Motorola GPS Receiver Technical Reference Manual*, pp. 4-11, 22-27, Motorola, October, 1993.

[MOTO93-2] *Motorola GPS Evaluation Kit Quick Start Guide*, pp. 2-1 - 4-17, A-1 - A-15, Motorola, June, 1993.

[MOYA93] Moya, D.C., and Elchynski, J.J., "Evaluation of the Worlds's Smallest Integrated Embedded GPS/INS, the H-764G", *The Institute of Navigation Proceedings of 1993 National Technical Meeting*, San Francisco, CA, January 20-22, 1993, pp. 275-286.

[NAGE92] Nagengast, S., *Correction of Inertial Measurements Using GPS Update for Underwater Navigation*, Master's Thesis, Naval Postgraduate School, Monterey, CA, pp. 4-6, March 1992.

[OSSE93] Osse, J. and Light, R., "Line Deployment by a Miniature AUV Under Arctic Ice", *Proceedings of the 8th International Symposium on Unmanned Untethered Submersible Technology*, September 27-29, 1993, pp. 277-290.

[PARK80] Parkinson, B.W., "Overview", *Global Positioning System*, Vol. 1, The Institute of Navigation, Washington, D.C., 1980, pp. 1-2.

[PREC94] Precision Navigation, *TCM1 Electronic Compass Sensor Module, Data Sheet*, Mountain View, CA, June, 1994.

[PARK80] "PX176 Series Pressure Transducer Data Sheet, pp. 1-2, Omega Technologies Limited, (no date).

[ROER93] Roer, M. and Jabert, J., "Proof of Concept: Research Using An Autonomous Underwater Vehicle (AUV)", *Proceedings of the 8th International Symposium on Unmanned Untethered Submersible Technology*, September 27-29, 1993, pp. 252-264.

[SOUE92] Souen, K., and Nishida, T., "The World's Smallest 8-Channel GPS Receiver", *Proceedings of The Institute of Navigation GPS-92*, Albuquerque, NM, September 16-18, 1992, pp. 707-713.

[STEV93] Stevens, C.D., *A Software Architecture for a Small Autonomous Underwater Vehicle Navigation System*, M.S. Thesis, Naval Postgraduate School, Monterey, CA, June, 1993.

[TUOH93] Tuohy, S.T., Patrikalakis, N.M., Leonard, J.J., Bellingham, J.G., and Chryssostomidis, C., "AUV Navigation Using Geophysical Maps with Uncertainty", *Proceedings of the 8th International Symposium on Unmanned Untethered Submersible Technology*, September 27-29, 1993, pp. 265-276.

[VAND80] Van Dierendonck, A.J., Russell, S.S., Kopitzke, E.R., and Birnbaum, M.,"The GPS Navigation Message", *Global Positioning System*, Vol. 1, The Institute of Navigation, Washington, D.C., 1980, pp. 55-73.

[WOOD85] Wooden, W. H., "NAVSTAR Global Positioning System: 1985", *Proceedings of the First International Symposium on Precise Positioning with the Global Positioning System*, April 1985, pp. 23-32.

[YOUN91] Youngberg, J.W., "A Novel Method for Extending GPS to Underwater Applications", *NAVIGATION, Journal of The Institute of Navigation*, Vol. 38, No. 3, Fall 1991.

INITIAL DISTRIBUTION LIST

1.	Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
2.	Dudley Knox Library Code 052 Naval Postgraduate School Monterey, CA 93943-5002	2
3.	Chairman, Code CS Computer Science Department Naval Postgraduate School Monterey, CA 93943-5000	1
4.	Dr Robert McGhee, Code CS/Mz Computer Science Department Naval Postgraduate School Monterey, CA 93943-5000	1
5.	Dr. James Clynch, Code OC/Cl Oceanography Department Naval Postgraduate School Monterey, CA 93943-5000	1
6.	Dr. Se-Hung Kwak, Code CS/Kw Computer Science Department Naval Postgraduate School Monterey, CA 93943-5000	1
7.	Dr. Randy Brill Naval Research and Development NCCOSC RDTE DIV 511 53420 Craig Rd. Rm 200 San Diego, CA 92152-6267	1
8.	Russ Whalen, Code CS Computer Science Department Naval Postgraduate School Monterey, CA 93943-5000	1

9. ADM Pearson
Commander, Mine Warfare Command
Naval Air Station Corpus Christi
Corpus Christi, TX 78419-5000 — 1

10. Guy Oliver
University of California, Santa Cruz
233 Northrop Place
Santa Cruz, CA 95060 — 1

11. Dr. James Eagle, Code UW/Er
Undersea Warfare Department
Naval Postgraduate School
Monterey, CA 93940-5000 — 1

12. Dr. Anthony J. Healey, Code ME/Hy
Department of Mechanical Engineering
Naval Postgraduate School
Monterey, CA 93940-5000 — 1

13. Donald Brutzman, Code OR/Br
Department of Operations Research
Naval Postgraduate School
Monterey, CA 93940-5000 — 1

14. LT Nancy A. Norton
1514 NE Beulah Drive
Roseburg, OR 97470 — 2